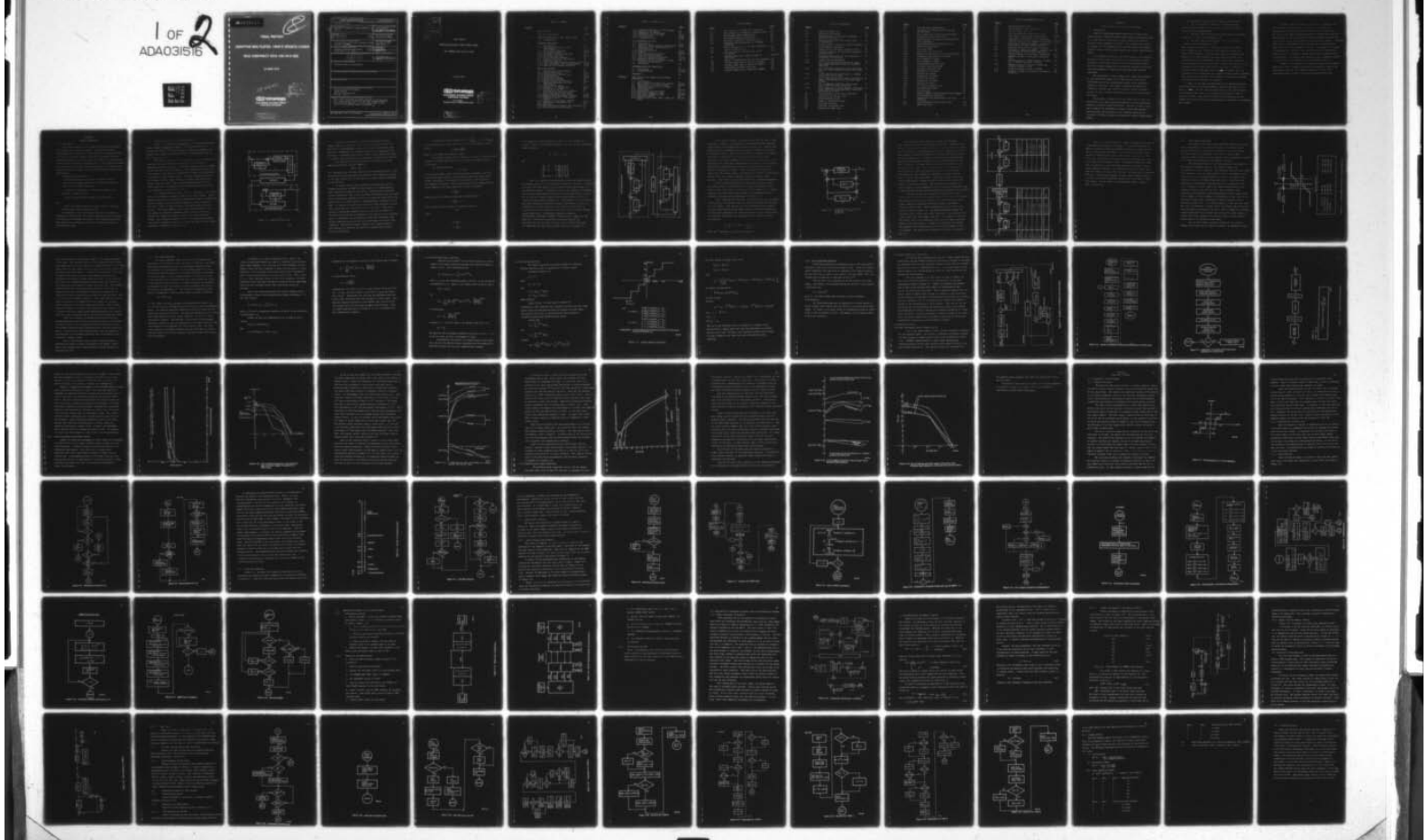
AD-A031 516 GTE SYLVANIA INC NEEDHAM HEIGHTS MASS ELECTRONIC SYS--ETC F/G 17/2
ADAPTIVE MULTILEVEL 16KB/S SPEECH CODER.(U)
JUN 76 A GOLDBERG DCA100-76-C-0002

UNCLASSIFIED NL

1 OF 2
ADA031516



FINAL REPORT

ADAPTIVE MULTILEVEL 16KB/S SPEECH CODER

DCA CONTRACT DCA-100-76-C-002

14 JUNE 1976

GTE SYLVANIA
INCORPORATED
ELECTRONIC SYSTEMS GROUP —
EASTERN DIVISION

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: DCA100-76-C-0002

2. GOVT ACCESSION NO.:

3. RECIPIENT'S CATALOG NUMBER:

4. TITLE (and Subtitle): ADAPTIVE MULTILEVEL 16 KB/S SPEECH CODER.

5. TYPE OF REPORT & PERIOD COVERED: Final Report. Nov 1975-June 1976

6. PERFORMING ORG. REPORT NUMBER:

7. AUTHOR(s): A. Goldberg

8. CONTRACT OR GRANT NUMBER(s): DCA100-76-C-0002

9. PERFORMING ORGANIZATION NAME AND ADDRESS:
GTE Sylvania, Electronics Systems Group
Eastern Division
77 "A" Street, Needham Heights, MA 02194

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:
P.E. No. 33126K
Task 15101

11. CONTROLLING OFFICE NAME AND ADDRESS:
Defense Communications Agency
Washington, D.C. 20305

12. REPORT DATE: 14 June 1976

13. NUMBER OF PAGES: 160

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): 165 p.

15. SECURITY CLASS. (of this report): Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT (of this Report)

Unlimited public distribution

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Adaptive Predictive Coding
Speech Coding
Digital Voice Terminal
Residual Coding

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report describes software designed by GTE Sylvania for a high quality 16,000 bit per sec. speech terminal. This software operates in a full duplex mode on two Sylvania Programmable Signal Processors.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED 406451

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

LB

FINAL REPORT

ADAPTIVE MULTILEVEL 16KB/S SPEECH CODER

DCA CONTRACT DCA-100-76-C-0002

14 June 1976

GTE SYLVANIA INCORPORATED
ELECTRONIC SYSTEMS GROUP
EASTERN DIVISION
77 "A" STREET
NEEDHAM HEIGHTS. MASSACHUSETTS 02194

TABLE OF CONTENTS

TABLE OF CONTENTS (cont.)

LIST OF TABLES

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (cont.)

LIST OF ILLUSTRATIONS (cont.)

SECTION I

Adaptive Multilevel 16 Kbps Speech Coder

1.0 Introduction

Under the eight month Adaptive Multilevel 16 Kbps Speech Coder contract, (DCA100-76-C-0002), GTE Sylvania developed software for a high quality 16,000 bit/sec speech transmission terminal having an audio bandwidth of 3000 Hz. This software was designed to operate in a full duplex mode on two GTE Sylvania Programmable Signal Processors (PSP) already owned by the Defense Communications Agency.

As a benchmark to evaluate the voice quality of the Adaptive Multilevel Speech Coder, GTE Sylvania also developed full duplex software for Continuous Variable Slope Delta Modulation (CVSD). This CVSD software enabled DCA to modify and optimize the internal operating parameters of CVSD and to experiment with new CVSD algorithms.

The development of these 16 Kbps voice coders was motivated by the expected deployment of a 16 Kbps secure voice network. A necessary part of any transmission network is the subscriber terminal or telephone. For digital communication network the transmitter terminal must convert the analog speech to digital, encrypt the data, receive incoming data, decrypt it and reproduce the speech.

Unfortunately, 16 Kbps is too low a data rate to send voice without the use of data reduction algorithms and these algorithms generally introduce audible distortion. The goal of these studies was to improve the performance existing 16 Kbps CVSD terminals and investigate the potential of new terminals employing the principles of Adaptive Predictive Coding with Adaptive Quantization (APCQ).

The significant results of this contract are threefold:

1. The voice quality of CVSD can be improved through relatively minor changes in the internal parameters or through the use of new CVSD algorithms.

2. The APCQ coder provides voice quality that is noticeably superior to the best CVSD algorithms, and

3. The APCQ coder can perform well even in the presence of channel errors exceeding 1 in 10^3.

These findings show that it is technically feasible to improve voice transmission at 16 Kbps over that obtained by using the present CVSD terminals. Using the first result, relatively minor changes in the CVSD telephone subscriber unit can provide voice transmission with less distortion. The second result, indicates that a new APCQ coder in the subscriber terminal could provide voice transmission with almost imperceptible distortion. Although the circuitry needed to implement these adaptive multilevel coders is considerably more complex than that needed for CVSD, studies performed under this contract indicate that largescale integration techniques can minimize this difference in complexity.

In fact, we show that the APCQ coder can be implemented around an LSI microprocessor using approximately $700 in parts and about 42 watts of power. Moreover, the processing speed of this microprocessor is sufficiently fast that it can analyze and synthesize 25 msec of speech in 16 msec. Thus 40% of the time the LSI microprocessor can perform other functions such as synchronization, encryption and telephone line control.

Of course, CVSD will always be less costly to implement than our design for the APCQ coder. Nevertheless, our design for implementing the complete subscriber terminal with the APCQ coder may cost no more than the present subscriber terminal employing CVSD. This equality in cost occurs because analog to digital conversion of voice is only one of many features of the subscriber terminal. Telephone line control, encypherment, and synchronization can significantly impact the cost of the subscriber terminal. Our design, based around a microprocessor, provides a cost effective method of implementing these features in software by sharing microprocessor hardware with the APCQ coder. The present CVSD terminal requires separate hardware for each feature, thus raising its total complexity and cost.

Because of the improved voice quality provided by the Adaptive Multilevel coders, further work should be performed to simplify the algorithms, to improve their hardware design, and to test their performance under realistic operating environments more fully.

SECTION 2

Software Simulations

2.0 Introduction

This study consisted of two parallel efforts for developing full duplex voice processing systems. The simulation studies investigated new techniques for coding speech and reducing the data rate and the real-time software development implemented the most promising techniques and several CVSD algorithms on the GTE Sylvania PSP for further evaluation and demonstration. This real-time software was delivered to the Defense Communications Agency for operation on their two PSP's built under a prior contract (DCA100-74-C-0058) with GTE Sylvania.

Specifically, GTE Sylvania

- Developed simulations of adaptive predictive coders with multilevel Quantizers(APCQ) at 16 Kbps using several forms of error signal quantizers.
- Studied the performance of these APCQ systems in the presence of channel errors
- Wrote full duplex APCQ software for the DCA PSP's.

and

- Wrote full duplex software for three forms of continuous variable slope delta modulation (CVSD)

The remainder of this section will describe the operating principles of APCQ and the error signal quantizers and will provide the results of the FORTRAN simulations. The following section will than describe the operation of the real-time CVSD coder and the real-time APCQ coder.

2.1 Operating Principles of the APCQ Terminal

Figure 2-1 illustrates the principles of a speech transmission based on adaptive predictive coding[1,2,3] with adaptive multilevel quantization (APCQ).[4,5] The APCQ technique estimates or predicts the present input samples s_n from past history of the waveform, that is $\hat{s}_n = P(s_{n-1}, s_{n-2}, \ldots)$.

The residual signal or error signal $e_n = s_n - \hat{s}_n$, along with the function P provides sufficient information for the receiver to regenerate the input precisely. In general, however, the error signal is quantized and is not sent exactly. This quantization distorts the output speech so the choice of an appropriate method for quantization is important to good quality speech. A properly designed predictor will produce an error signal having less dynamic range and smaller variance than the input. Thus a quantizer operating on the error signal needs fewer bits/sample than one operating on the input, and this significantly lowers the data rate.

The optimum predictor depends upon the statistics of the input and thus the predictor parameters adapt as the input changes. These parameters are chosen to minimize the mean-squared error e_n between the predicted samples $\hat{s}_n$ and the actual samples s_n, over an analysis interval or frame of length T which is typically between 20 and 25 ms long. This frame length is often called the band length. Once calculated, the predictor parameters remain fixed for the entire interval but change from frame to frame as the input statistics change. Thus, during each analysis frame, the algorithms determine the predictor parameters from the input data and form the error signal sequence by filtering the input digitally.

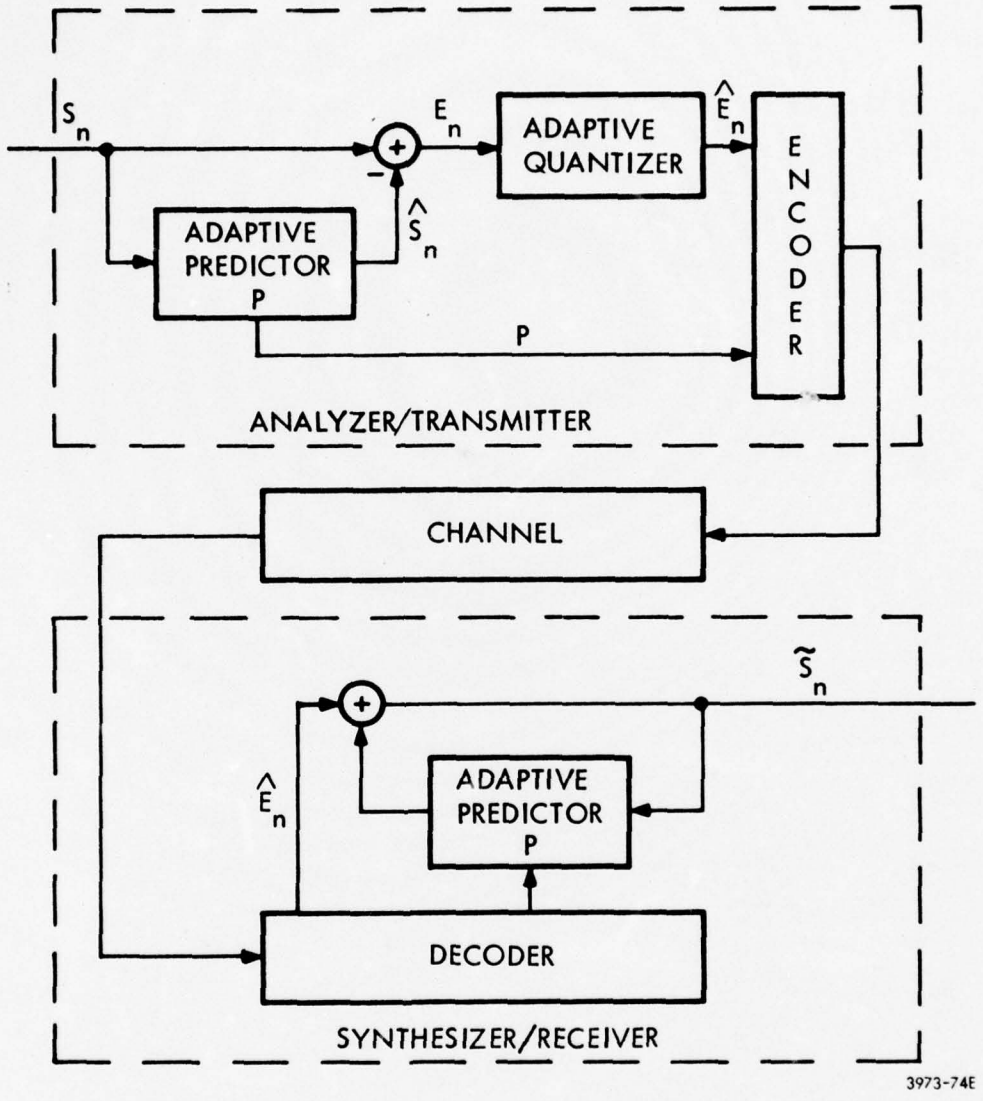


Figure 2-1. Simplified APCQ Coder

For speech, the predictor function P was broken into two separate smaller predictors P1 and P2, as illustrated in Figure 2-2. The form of P1 acknowledges that speech is often quasi-periodic with period M. Consequently, P1 estimates speech as $\hat{s}_n = \alpha s_{n-M}$ where the pitch gain α indicates that there is either a gain change from period to period or that the speech is not perfectly correlated with period M.

For calculation of the pitch period M we chose the average magnitude difference function (AMDF)(6)

$$\mathrm{AMDF}(j) = \sum_{n=1}^{T} |s_n - s_{n-j}|$$

as a computationally efficient method for its estimation pitch M. This function has no multiplications and requires little numerical scaling. It also has a sharp null for that value of j which usually corresponds to the period M.

In speech, not all pitch periods are possible and thus the terminal calculates the AMDF for those values of j corresponding to pitch frequencies between 70 and 340 Hz. Accurate pitch extraction requires additional logic to test for false or multiple nulls, but since later processing can partially correct for occasional pitch errors, the period M is set to that value of j corresponding to a minimum value of AMDF. To speed up the processing further, the AMDF calculation in the APCQ system sums over every third value of the input. This causes spectral aliasing and occasional shifts in nulls in the AMDF, but listening tests indicate no impairment in the quality of the resulting speech. Furthermore, in calculating the AMDF, the algorithms form partial summations which are then scaled and added to form the final summation. Overflows are always clamped to the largest number of the processor but this does not affect the minimum values needed for pitch extraction.

To minimize the total squared error $E = \sum_{n=1}^{T} (s_n - \alpha s_{n-M})^2$, we can differentiate E with respect to α and set the result to zero giving

$$\alpha = \sum_{n=1}^{T} s_n s_{n-M} / \sum_{n=1}^{T} s_n^2 - M$$

where T = the frame baud length in samples.

For periodic sounds, such as vowels, α is close to unity, but for noiselike consonants which have little correlation M samples apart, α is near zero.

The reduced waveform,

$$v_n = s_n - \alpha s_{n-M},$$

or first error signal still contains sufficient redundancy such that a second Nth-order predictor P2 can further reduce the output signal energy, especially if the speech is not periodic or if the period is estimated incorrectly. This second predictor uses a weighted sum of N past samples of the speech waveform to form the estimate

$$\hat{v}_n = \sum_{i=1}^{N} a_i v_{n-i}$$

where the a_i's are chosen to minimize the squared error

$$U = \sum_{n=1}^{T} (v_n - \hat{v}_n)^2$$

and are given by the solution to the matrix equation

$$\phi a = C$$

where

$$\phi_{ij} = \sum_{n=1}^{T} v_{n-i} v_{n-j}$$

$$C_j = \sum_{n=1}^{T} v_n v_{n-j}.$$

If we window the reduced waveform so that it is zero outside the interval $1 \leq n \leq T$ (stationarity assumption), we have the autocorrelation normal equations

$$\phi_{ij} = \sum_{n=1}^{T-(i-j)} v_n v_{n-(i-j)} = R_{i-j}$$

and

$$\begin{bmatrix} R_0 & R_1 & \cdots & R_{N-1} \\ R_1 & R_0 & \cdots & R_{N-2} \\ & & & \cdot \\ & & & \cdot \\ & & & \cdot \\ R_{N-1} & R_{N-2} & \cdots & R_0 \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ a_2 \\ \cdot \\ \cdot \\ \cdot \\ a_N \end{bmatrix} = \begin{bmatrix} R_1 \\ R_2 \\ \cdot \\ \cdot \\ \cdot \\ R_N \end{bmatrix}.$$

This is a symmetric Toeplitz matrix because the elements along any diagonal parallel to the principal diagonal are identical. Efficient solutions exist that supply the a_i's and the mean-square energy U in the difference signal.[7] In addition, because the elements of the matrix arise from an autocorrelation function, the stationary matrix solution for the a_i's will yield a stable filter during synthesis, with the recursive filters shown in Figure 2-2. Unfortunately, the a_i's do not make good transmission parameters because quantization or errors in transmission can cause the poles of the receiver filter given by $1/(1-P1)(1-P2)$ to move outside the unit circle in the Z plane, causing unstable waveforms. Consequently, auxiliary parameters called partial correlation (PARCOR) coefficients K_i (which are the negatives of reflection coefficients discussed by Atal[8]) are calculated from the a_i's and, as long as these PARCOR coefficients have magnitude less than unity, system stability is assured.[6,7]

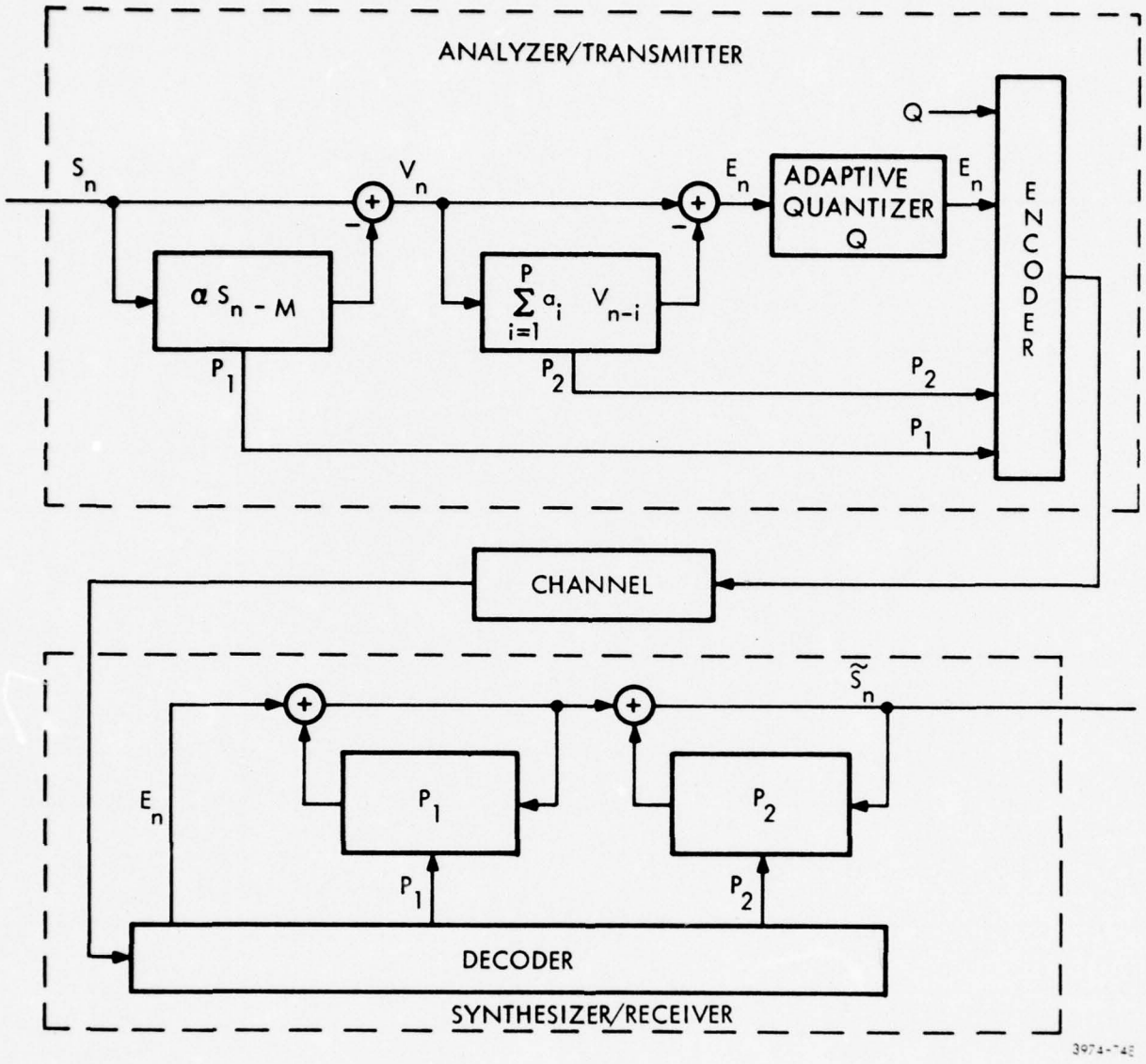


Figure 2-2. Simplified APCQ System

In the actual terminal, instead of quantizing the error signal as Figure 2-2 depicts, the analyzer is reformulated and the quantizer placed inside the filtering loop as shown in Figure 2-3. Without this quantizer, this configuration has the transfer function (1-P1)(1-P2) as discussed earlier. With this new formulation, however, we eliminate the effects of small errors in accumulation caused by the error quantizer. This occurs because the only input to the predictor is the quantized error signal and, consequently, the predictor at the analyzer generates the same predicted sequence as the receiver does in the absence of transmission errors. Since it compares this predicted signal with the original input, the analyzer can correct for distortions caused by the error signal quantizer, altering the error signal sequence. This would not be possible without the feedback arrangement.

There are numerous methods for quantizing the error signal, and Jayant provides a good discussion of adaptive techniques which allow at least four levels or two bits per error sample (12). A major portion of this study effort was devoted to investigating various quantizers and these will be discussed in Section 2.2 .

After computing the pitch gain α, the period M, and the PARCOR coefficient K_i, the analyzer digitally filters the speech and quantizes the error signal. Then the K_i's, α, M, the quantized error sequence e_n, and a normalization factor NORM, needed to perform the operations in fixed-point arithmetic, are quantized and sent to the receiver where the predictor coefficients are regenerated in an iterative fashion by computing

$$a_j^{(i)} = a_j^{(i-1)} - a_{i-j}^{(i-1)} * K_i \qquad j = 1,2,3,\cdots,i-1,$$
$$a_i^{(i)} = K_i \qquad i = 1,2,3,\cdots N$$

where $a_j^{(i)}$ represents a_j on the ith iteration.

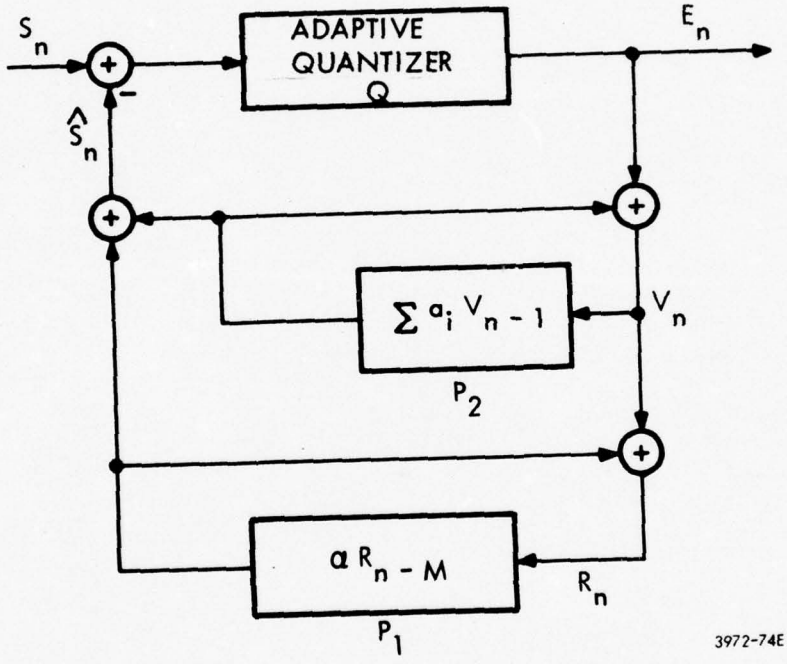
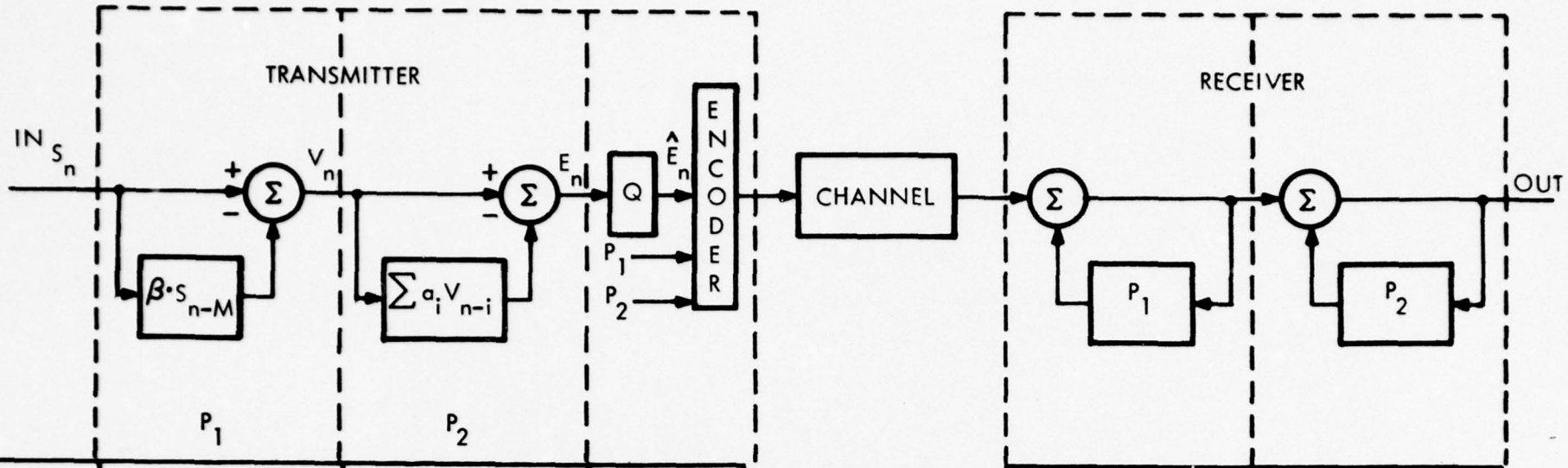


Figure 2-3. Reconfiguration of Analyzer to Minimize Effects of Quantizer

The synthesizer then creates an output time waveform that is both intelligible and pleasing to listen to. Moreover, this output is reasonably insensitive to errors in pitch extraction because the P2 predictor on the reduced waveform and the quantization of the error signal can partially compensate for wrong pitch values used in the first predictor P1. In fact, if the pitch period M incorrectly doubles, as often happens in practice, then predictions made by the first predictor P1 are made from two periods before instead of the previous period. This is not a serious error. If other incorrect values for M are chosen then different values of α and PARCOR coefficients are also computed to compensate, in part, for this error. Finally, the error signal, even though coarsely quantized, carries considerable information about the true pitch, should this pitch be incorrectly measured. Thus the APCQ speech process can function without severe degradation in noisy acoustic environments and on many different speakers where accurate pitch extraction is difficult.

The APCQ process can be modified to obtain other forms of speech coders currently being studied and discussed in the literature. These modification can yield Adaptive Pulse Code Modulation Coders (APCM), Adaptive Differential PCM Coders (ADPCM), and Adaptive Predictive Coders (APC). These configurations for a 16 Kb/s voice coder are shown in Figure 2-4. The major differences between the different forms of coders are in the complexity of the predictor and in the form of the error signal quantizer. For example, in Adaptive PCM (APCM), there is no prediction, only the error signal quantizer adapts to the input speech. In Adaptive Differential PCM (ADPCM) the predictor uses a weighted value of the previous sample to form its estimate. The weight never changes and its value is typically near 0.9.



Coder Technique	Pitch Predictor	Coefficient Predictor	Adaptive Quantizer (8-level)	Inverse Pitch Predictor	Inverse Coeff. Predictor
APCM	No	No	Yes	No	No
ADPCM	No	Yes-one fixed pred. coeff.	Yes	No	Yes-one fixed pred. coeff.
APCQ-j B≠0	Yes	Yes-j adapt. pred. coeff.	Yes	Yes	Yes-j adapt. pred. coeff.
APCQ-j B=0	No	Yes-j adapt. pred. coeff.	Yes	No	Yes-j adapt. pred. coeff.
LPC-4*	Yes	Yes-4 adapt. pred. coeff.	No**	Yes	Yes-4 adapt. pred. coeff.

**Fixed 2-level

*Below 8Kb/s

Figure 2-4. Relationships Among 16 Kb/s Speech Coder Configurations

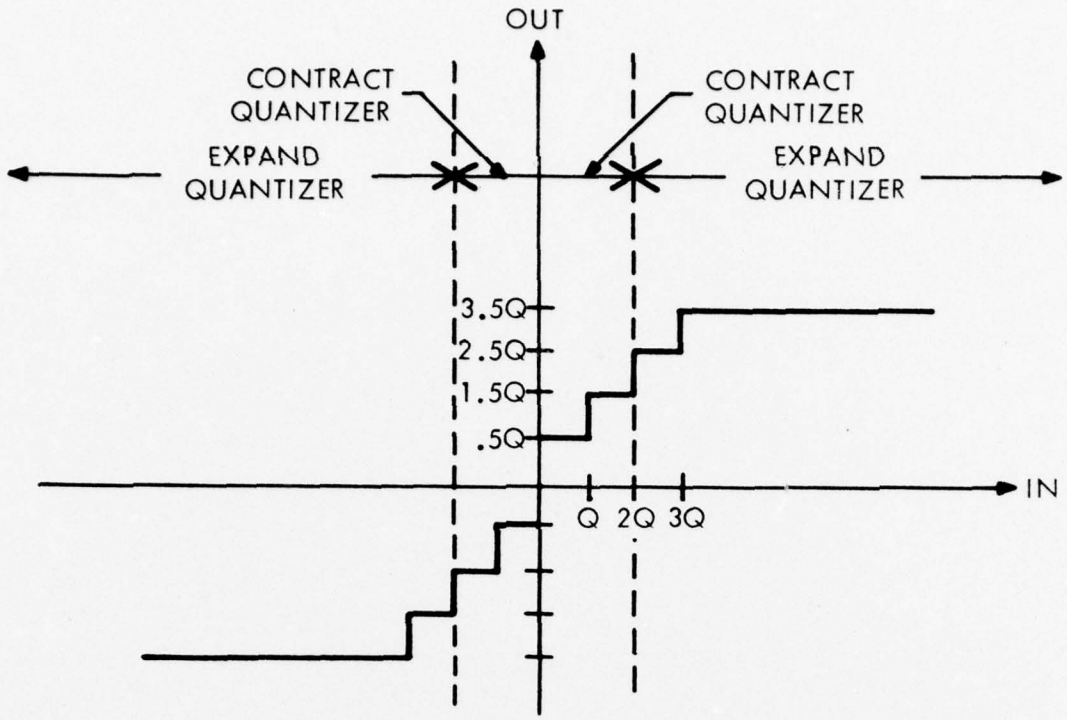
There is no pitch prediction. Again, an adaptive quantizer is incorporated which adapts to the input. As already discussed, the adaptive predictive coder with adaptive quantization (APCQ) has two fairly sophisticated predictions: the coefficient predictor with typically 4 weights and a pitch predictor with pitch gain α. The pitch loop can be easily eliminated without reconfiguring the coder by setting the pitch gain α to zero. The Adaptive Predictive Coder (APC) is identical to the APCQ coder except for the form of error signal quantizer employed. Its quantizer has only two levels whose amplitude changes once per frame interval. Its data rate is typically 6400 bps. The APCQ coder employs an error signal quantizer having 3 or more levels whose amplitudes change either once/frame of after each sample. The data rate is correspondingly higher, typically 9600 to 16,000 bps.

2.2 Error Signal Quantizers

The design of the error signal quantizer significantly affects the voice quality of the synthesized speech. Under this contract we have investigated several types of quantizers but considerably more work needs to be performed in this area to find a simple quantizer which performs well with channel errors and which will permit us to reduce the complexity of the adaptive predictor.

The significant feature of all the quantizers investigated is the adaptively changing size, Q. For example, Figure 2-5 illustrates an 8 level quantizer. This quantizer takes an input signal and converts it to one of eight discrete levels. The number of this level (000 to 111 in binary notation) is sent to the receiver which converts it back to the desired amplitude value. Since the curve describing the input/output characteristics of the quantizer looks like a staircase of step high Q (for uniform quantization), Q is commonly referred to as the step size. Until recently, these quantizers were memoryless, and once calculated, Q never changed even if the signal statistics changed. Unfortunately, if the input to the quantizer becomes large, the quantizer tends to saturate and not describe the input well. In a similar fashion, a very low level input may force the quantizer to always output its minimum step size and not track the small variations in the input. What is needed is a quantizer having memory which will increase the step size for large signals or signals with large variance and decrease it when the input signals become small or the variance becomes small.

All the error signal quantizers studied in this contract changed their step sizes at regular intervals. By changing the step



DESIGN EQUATIONS

1.	$\|IN\| \leq Q$;	$Q \rightarrow .9Q$	OUT = .5Q SGN
2.	$Q < \|IN\| \leq 2Q$;	$Q \rightarrow .9Q$	OUT = 1.5Q SGN
3.	$2Q < \|IN\| \leq 3T$;	$Q \rightarrow 1.25Q$	OUT = 2.5Q SGN
4.	$3Q < \|IN\|$;	$Q \rightarrow 1.75Q$	OUT = 3.5Q SGN

Figure 2-5. 8-Level Adaptive Quantizer (Jayant Quantizer)

sizes in this manner, the quantizer adapts to the changing variance of its input. Two distinct classes of these quantizers exist; those that change their step size based on the transmitted value of the error signal and those that change their step size based on the variance of the unquantized error signal. P. Noll[9,10] has called the first form "backward" quantizers since they look backward over previously quantized error samples to adjust their step size. He has labelled the second form "forward" quantizers because they look forward over the unquantized error sample to obtain their step size. The backward quantizers need not send the quantizer step size to the receiver because the receiver can regenerate this value by looking at the transmitted sequence representing the quantized error waveform. This is not true of the forward quantizers. Here, since the step size is based on the value of the unquantized error signal, the receiver cannot regenerate it from the transmitted sequence. Thus, forward quantizers transmit the value of the quantizer to the receiver. At a constant data rate, coders having forward quantizers have slightly lower audio input bandwidth than do those having backward quantizers.

Under this contract, we determined that both forward and backward quantizers can be made to work equally well in the presence of channel errors. This contradicts the work of Noll who believed that forward quantizers which transmitted their step size would work better. At the time of his writing, however, Noll was unaware of modifications to the backward quantizers which would improve their tolerance to channel errors.

We will first discuss these backward quantizers and the modifications needed to improve performance with channel errors. Then we will describe the forward quantizers and the necessity to send the step size.

2.2.1 The Jayant Quantizer

The Jayant quantizer adapts its step size instantaneously, expanding or contracting its value each time it quantizes a new sample. Figure 2-5, illustrating an 8 level (3 bit/sample) quantizer, shows the operation of this quantizer. If the incoming sample is in one of the inner quantizing levels, the quantizer step size is decreased for the next sample. If the incoming sample is in one of the outer levels, the step size is increased instead. These changes are based on the quantized output and thus, the Jayant quantizer is classified as backward for there is no need to send the quantizer step size as the receiver can regenerate it. The constantly changing step size, which is attempting to track the input signal variance is given by

$$q_k = M(I_k)\, q_{k-1}$$

at time k where the $M(I_k)$ represent the multiplication factors for the step size on the kth sampling interval and I_k is the kth transmission symbol. The adaptive quantizer formulation need not be restricted to quantizers having only 8 levels. Any quantizer having more than 2 levels can be made adaptive with the Jayant algorithm. In fact, a 3 level quantizer is possible. This quantizer, which uses (1.6 bits/sample) quantizer, expands the step size whenever its two outer values are used and decreases the step whenever the inner value is used. The three level quantizer is desirable whenever the data rates must not exceed 9600 bps, whereas the 8 level quantizer requires at least 16000 bps at 2500 Hz bandwidth.

In general, for a given transmission rate, there is an inverse relationship between the number of quantizer levels and the audio input bandwidth. At 16,000 bps an 8 level quantizer (3 bits/sample) limits the audio bandwidth to about 2500 Hz, while a 5 level quantizer (2.33 bits/sample) would permit about 3200 Hz bandwidth and a 4 level quantizer would allow more than 3300 Hz. Thus, using a 5 or 6 level quantizer instead of an 8 level quantizer, the audio bandwidth of the coder will increase from 2500 Hz but the quantizing noise will also increase because fewer quantizing levels are employed.

The Jayant algorithm performs well without channel errors. Unfortunately, channel errors significantly impair performance. To see this consider

$$q_k = M(I_k)\, q_{k-1} = \prod_{i=1}^{k} M(I_i)\, q_o$$

where I_i is the ith transmitted character and $M(I_i)$ is the multiplier corresponding to I_i.

Assume now that one transmission error is made at time ℓ causing

$M(I_\ell)$ to become $M(I_\ell')$

and

q_ℓ to become $q_\ell' = M(I_\ell')\, q_{\ell-1}$

afterwards the new quantizer levels for time greater than ℓ becomes

$$q_k' = \prod_{\substack{L=1 \\ L \neq \ell}}^{k} M(I_i)\, M(I_\ell')\, q_o \cdot \left[\frac{M(I_\ell)}{M(I_\ell)}\right]$$

or simplifying we obtain

$$q_k' = q_k \frac{M(I_\ell')}{M(I_\ell)}$$

Consequently, the effects of even a single channel error never die out. In fact, an error causes a level shift in q_k by the factor $M(I_\ell')/M(I_\ell)$. Simulations and real-time implementation show that errors cause noticeable fades and increases in output level. Only when the quantizer saturates at its maximum level or cuts off at its minimum level does the quantizer again track properly. Thus, the Jayant algorithm must be altered if it is to be useable over real communication channels.

2.2.2 The Modified Jayant Algorithm

Wilkinson and Goodman[11] have recently shown how a simple change to the Jayant algorithm can make this algorithm robust to channel errors. Their modification was

$$q_k = M(I_k)q_{k-1} \quad \alpha = \prod_{i=1}^{k} M(I_i)^{\alpha^{k-i}} q_o$$

where I_i is the ith transmitted symbol and $M(I_i)$ is the multiplier corresponding to I_i. Again if one channel error curves at time ℓ

$$M(I_\ell) \rightarrow M(I_\ell')$$

and

$$q_k' = \prod_{\substack{i=1 \\ i=\ell}}^{k} \left[M(I_i)\right]^{\alpha^{k-i}} M(I_\ell')^{\alpha^{k-\ell}} q_o \cdot \left[\frac{M(I_\ell)}{M(I_\ell)}\right]^{\alpha^{k-\ell}}$$

or simplifying

$$q_k' = q_k \cdot \left[\frac{M(I_\ell')}{M(I_\ell)}\right]^{\alpha^{k-\ell}}$$

Assuming $\alpha < 1$, then for times k far removed from ℓ ($k >> \ell$).

$$q_k' \rightarrow q_k$$

The speed of this convergence depends on how near α is to 1. If ℓ is not too close to unity, convergence is rapid.

Consequently, the effects of a single channel error decay with time for the modified Jayant algorithm and this algorithm is much more suitable for use over communications channels.

2.2.3 The Forney Quantizer

The Forney quantizer[5] was another attempt at creating a backward quantizer which is insensitive to channel errors.

As shown in Figure 2-6,

$$q_k = 2^{\varepsilon k}$$

where

$$\varepsilon_k = G_k + C_k$$

and

$$G_k \text{ is } \alpha G_{k-1} + M_1(I_k)$$

$$C_k = \beta C_{k-1} + M_2(I_k)$$

where specific

$M_1(I_k)$ $M_2(I_k)$ α & β are given in Figure 2-6.

Consequently, this quantizer also increases its step size when large inputs occur and contrasts the step size during low level inputs. The quantizer step size is non-uniform and reacts rapidly to occasional large spikes caused by pitch pulses.

Noting that

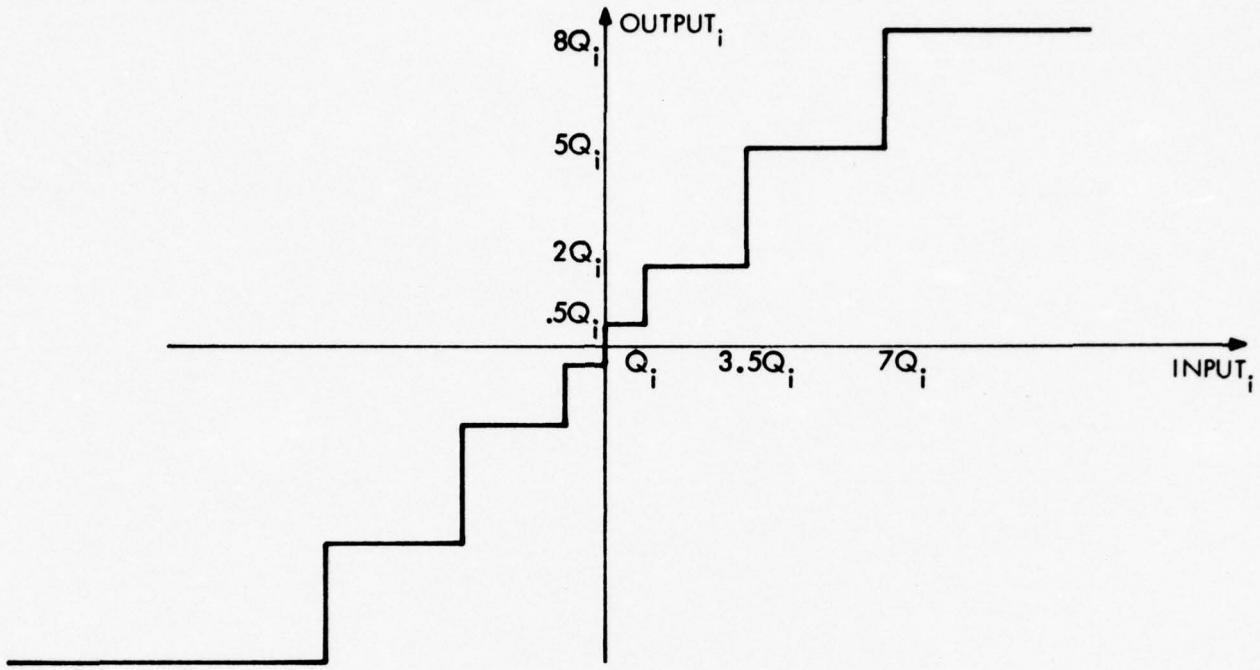
$$G_k = \sum_{i=1}^{k} \alpha^{k-i} M_1(I_i)$$

and

$$C_k = \sum_{i=1}^{k} \beta^{k-i} M_2(I_i)$$

we have

$$q_k = 2\left(\sum_{i=1}^{k} \alpha^{k-i} M_1(I_i) + \sum_{i=1}^{k} \beta^{k-i} M_2(I_i)\right)$$



$$Q_i = 2^{E_i}$$

$$E_i = G_i + C_i$$

AND

$$G_i = .99G_{i-1} + \begin{cases} 9/32 & \text{IF } |OUTPUT_{i-1}| = 8L_{i-1} \\ 5/64 & \text{IF } |OUTPUT_{i-1}| = 5L_{i-1} \\ 1/128 & \text{IF } |OUTPUT_{i-1}| = 2L_{i-1} \\ -1/16 & \text{IF } |OUTPUT_{i-1}| = 0.5L_{i-1} \end{cases}$$

$$C_i = .75C_{i-1} + \begin{cases} .875 & \text{IF } |OUTPUT_{i-1}| = 8L_{i-1} \\ 0 & \text{IF } |OUTPUT_{i-1}| = 5L_{i-1} \\ 0 & \text{IF } |OUTPUT_{i-1}| = 2L_{i-1} \\ 0 & \text{IF } |OUTPUT_{i-1}| = 0.5L_{i-1} \end{cases}$$

CHARACTERISTIC: SLOWLY ADAPTIVE EXCEPT DURING OCCURANCE OF LARGE PULSES. NO NEED TO TRANSMIT Q_i

8524-75E

Figure 2-6. Forney Adaptive Quantizer

at time ℓ assume a channel error causes

$$M_1(I_\ell) \rightarrow M_1(I_\ell')$$

$$M_2(I_\ell) \rightarrow M_2(I_\ell')$$

then

$$q_k' = 2\left(\sum_{\substack{i=1 \\ i\neq\ell}}^{k} \alpha^{k-i}M_i(I_i) + \alpha^{k-\ell}M_1(I_\ell') + \Sigma\beta^{k-i}M_2(I_i) + \beta^{k-\ell}M_2(I_\ell')\right)$$

by adding and subtracting

$$\alpha^{k-\ell}M_1(I_\ell) \text{ and } \beta^{k-\ell}M_2(I_\ell)$$

to the exponent

we have

$$q_k' = q_k \cdot 2^{\alpha^{k-\ell}\left(M_1(I_\ell') - M_1(I_\ell)\right)} \cdot 2^{\beta^{k-\ell}\left(M_2(I_\ell') - M_2(I_\ell)\right)}$$

as $k >> \ell$ $\alpha^{k-\ell} \rightarrow 0$

for $\alpha < 1$

and $q_k' \rightarrow q_k$

Thus the Forney quantizer also is insensitive to channel errors because after a channel error the step size approaches the desired step size with time. Moreover, this algorithm responds more rapidly to sudden changes in the input than does the modified Jayant algorithm.

2.2.4 The Fixed/Frame Quantizer

Fixed/frame quantizers as originally used in APC algorithms[1,2] calculate a step size for an entire frame of data. Since the receiver cannot regenerate this data from the quantized error signal sequence, the step size must be sent to the receiver once each frame. This is a forward type of quantizer as described by Noll.[9,10]

The step size Q is a function of the RMS energy in the input signal. The energy U is calculated during estimation of the predictor coefficients and

$$Q = 1/2(U/T)^{1/2}$$

where T = the frame length and the factor 1/2 was determined experimentally.

The fixed/frame quantizer is insensitive to errors because a single channel error effects only one value of the reconstructed error signal. Of course, if an error occurs in transmissing Q then an entire frame of data is shifted in level. Fortunately, the following frames of data are unaffected.

2.3 Fortran Fixed-Point Simulations

The various APCQ configurations and error signal quantizers were investigated through FORTRAN fixed-point simulations intended to implement the algorithms with the same numerical accuracy as the real-time software. Writing these codes in FORTRAN, permitted us to investigate many designs rapidly without the long debugging and check out times associated with writing real-time software.

The block diagram of the APCQ coder is shown in Figure 2-7 with a more detailed indication of the processing at the analyzer (transmitter) shown in Figure 2-3. Figure 2-8 indicates the general flow of data in the analyzer with Figure 2-9 indicating the data flow of the synthesizer (receiver). These flowcharts indicate that the speech stored on digital tape is read in a frame at a time, processed by the fixed-point algorithms, and reconstructed and placed on an output tape for listening. The I/O is handled by the digital tape handlers written by GTE Sylvania prior to this contract. Thus the I/O for the simulation was straightforward, and effect was placed on the scaling and normalization needed to perform the processing operations in fixed-point arithmetic. Appendix A provides a complete listing of the fixed-point simulation software with flow charts indicating the scaling and normalization operations in each arithmetic function.

2.4 Simulation Results

2.4.1 Coder Performance without Channel Errors

All of the speech coders of this contract attempted to preserve the time waveform shape. As a result, a reasonable performance measure for these systems is signal-to-noise ratio (S/N) as defined in Figure 2-10 Informal listening tests of these coders indicate that listeners generally prefer those systems having higher S/N ratios over those having lower ratios. The computer simulations developed S/N

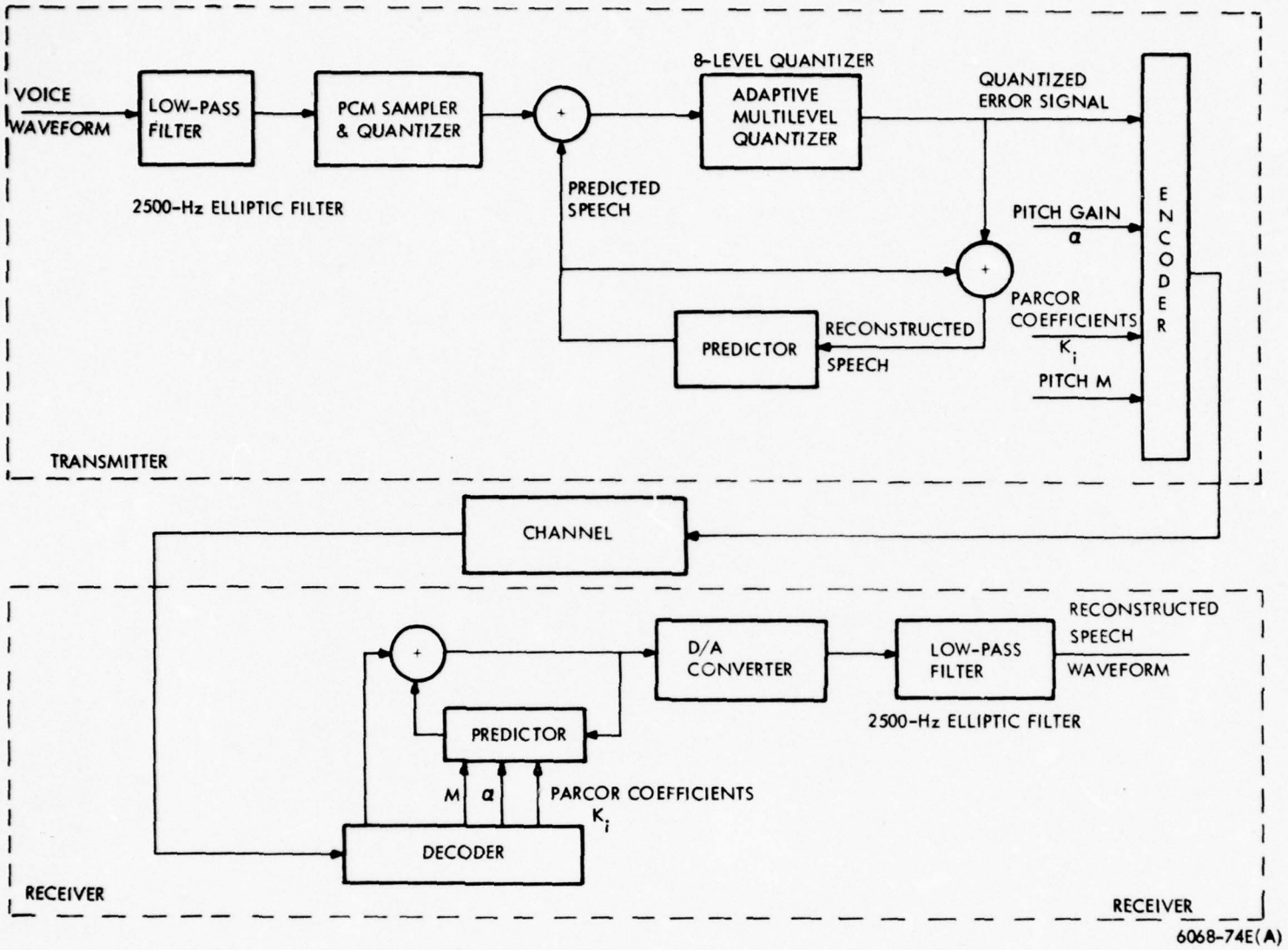


Figure 2-7. Simulation of APCQ Speech Encoding System

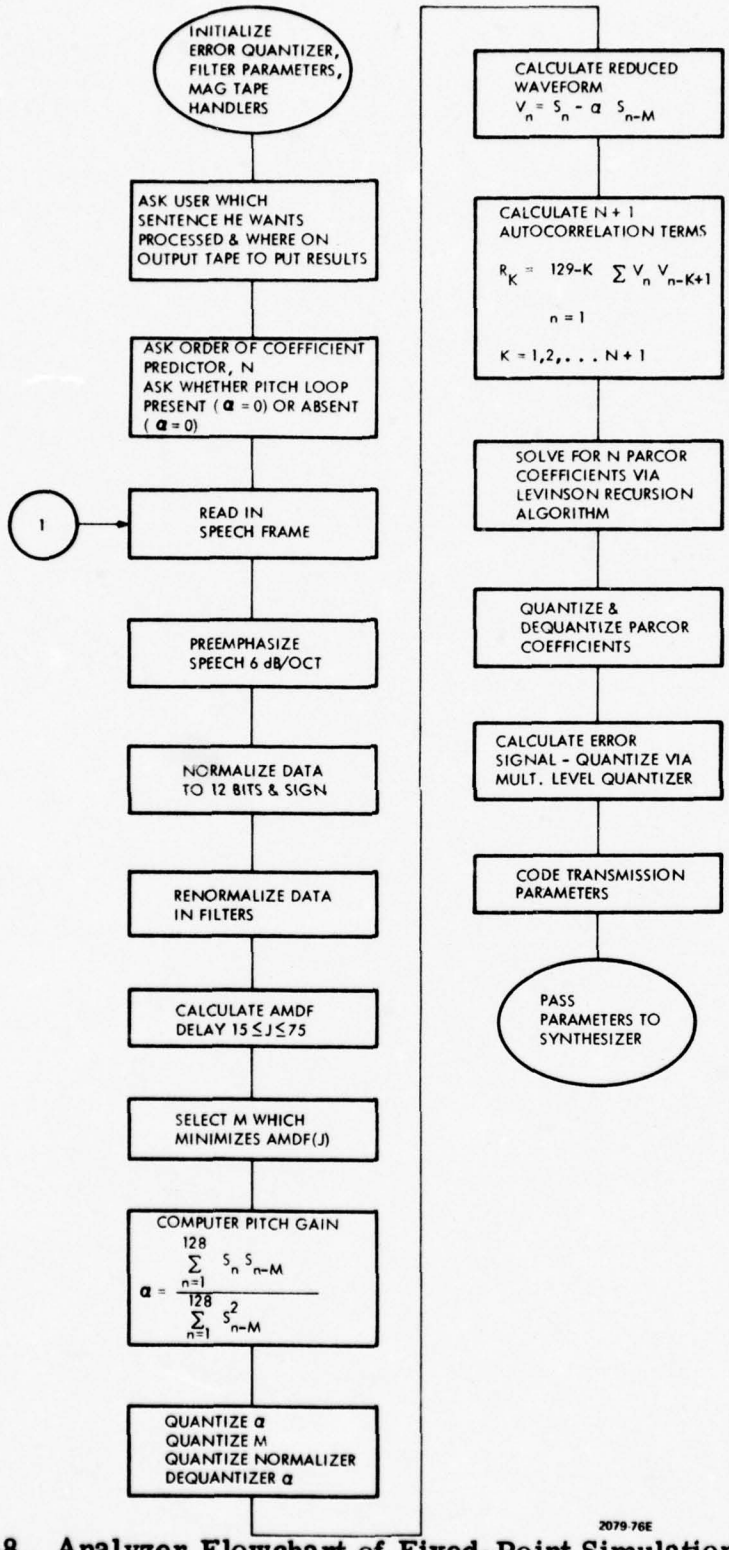


Figure 2-8. Analyzer Flowchart of Fixed-Point Simulation for APCQ Coder

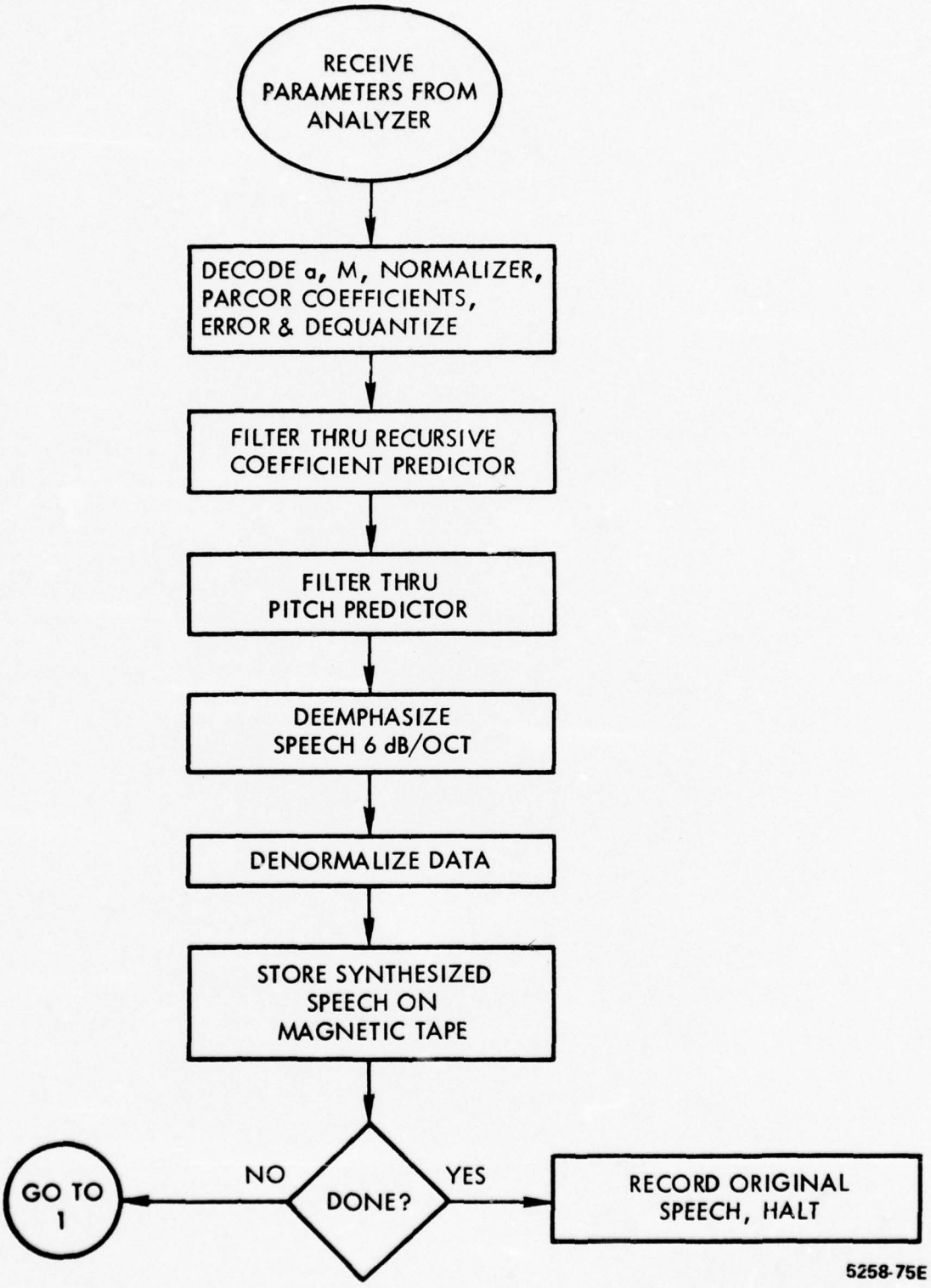
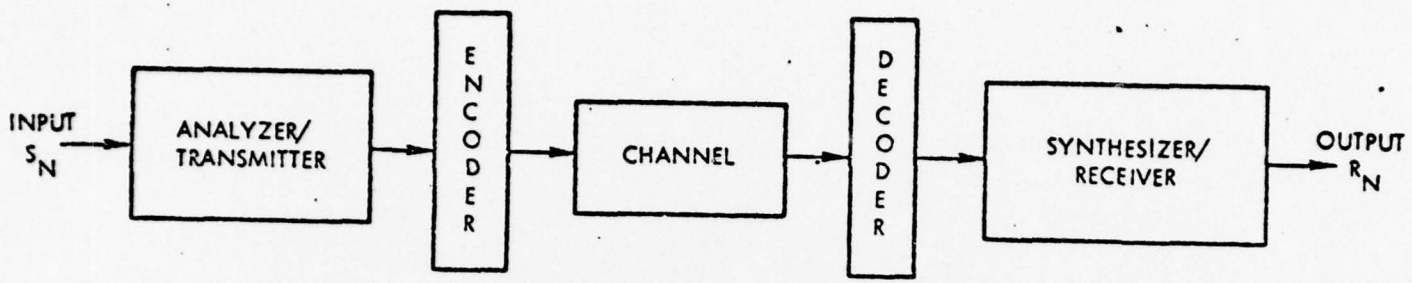


Figure 2-9. Synthesizer Flow Chart of the Fixed-Point Simulation for APCQ Coder



SIGNAL = S_N

NOISE = $S_N - R_N$

$$\text{S/N RATIO (IN dB)} = 10 * \text{LOG}_{10} \frac{\Sigma S_N^2}{\Sigma (S_N - R_N)^2}$$

5249-75E

Figure 2-10. Signal-to-Noise Measurement

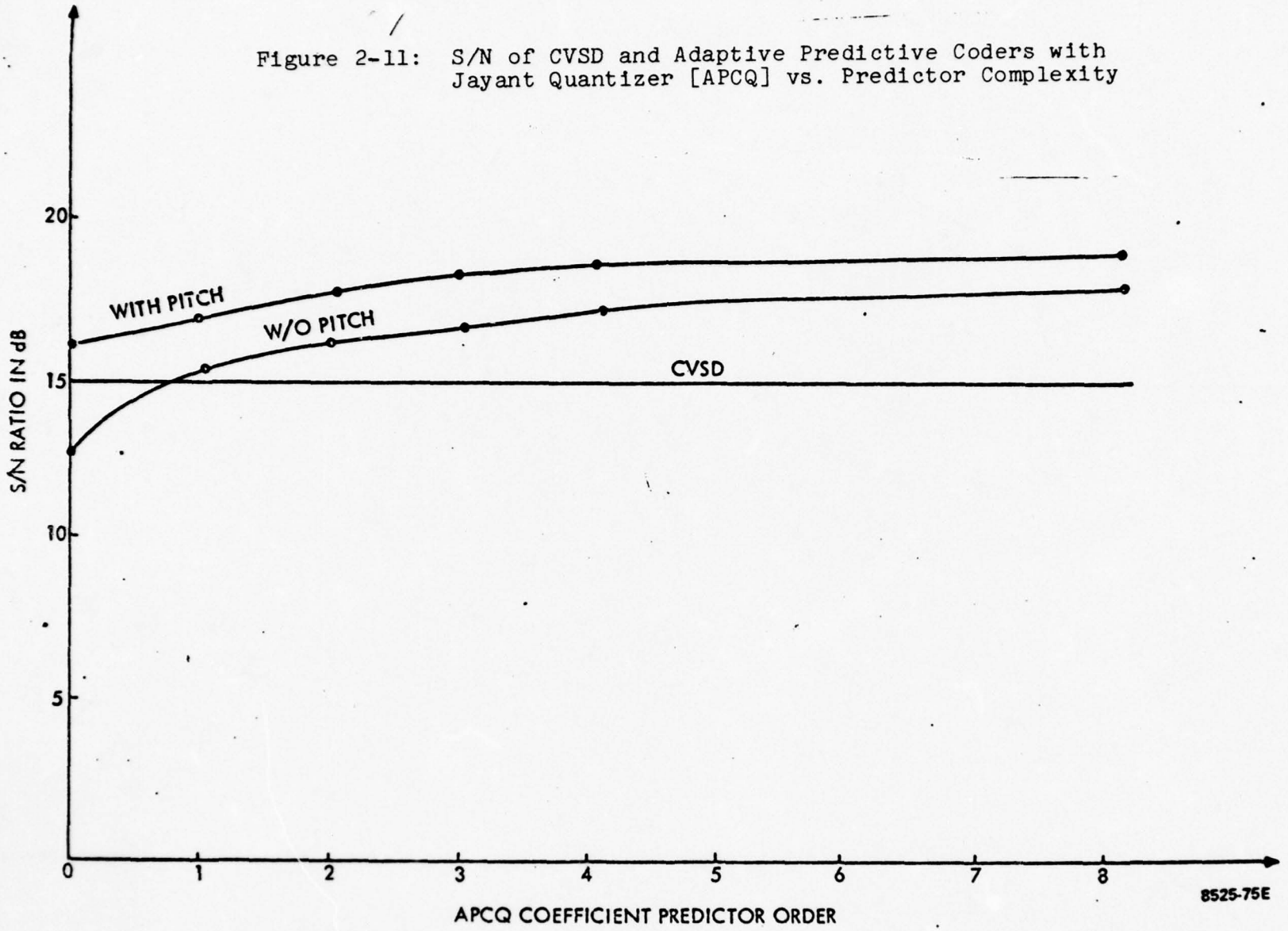
ratios for each non-overlapping 25 msec speech segment. These ratios were then averaged to obtain a cumulative S/N ratio for an entire sentence. It is these cumulative S/N ratios that are used in our results presented in the graphs of Figures 2-11 through 2-12.

Figure 2-11 shows the performance of APCQ systems using an 8 level Jayant quantizer operating at 16 kbps. This data was obtained by averaging the cumulative S/N ratio obtained for each sentence over a total of six input test sentences containing both male and female voices. Each test sentence contained about 100 analysis frames and thus, each point on Figure 2-11 represents the average of 600 frames. Referring to Figure 2-11, the performance increases with increasing predictor order and with the inclusion of a pitch loop. The zeroth order predictor without pitch is equivalent to APCM. This system performs roughly 2-3dB worse than the CVSD system we have chosen as our benchmark. Other simulations performed as part of this study indicate that ADPCM has a S/N equivalent to APCQ-1 (APCQ with 1 predictive coefficient) without pitch. As Figure 2-11 indicates, APCQ-1 (without pitch) is roughly equivalent to CVSD. The addition of a pitch loop adds between 1.5 and 2.0 dB to the S/N performance.

2.4.2 Coder Performance with Channel Errors

Figure 2-12 shows the performance of these coders in the presence of channel errors for the Jayant, Modified Jayant, Forney and fixed frame quantizers. Controlled random bit errors were applied to all transmitted data equally. The coder used to evaluate these quantizers was a first order adaptive predictor (APCQ-1) without a pitch loop. In preparing the data of this Figure, only one test sentence was used due to large amount of processing involved. Using selected points taken from other test sentences showed similar trends in performance.

Figure 2-11: S/N of CVSD and Adaptive Predictive Coders with Jayant Quantizer [APCQ] vs. Predictor Complexity



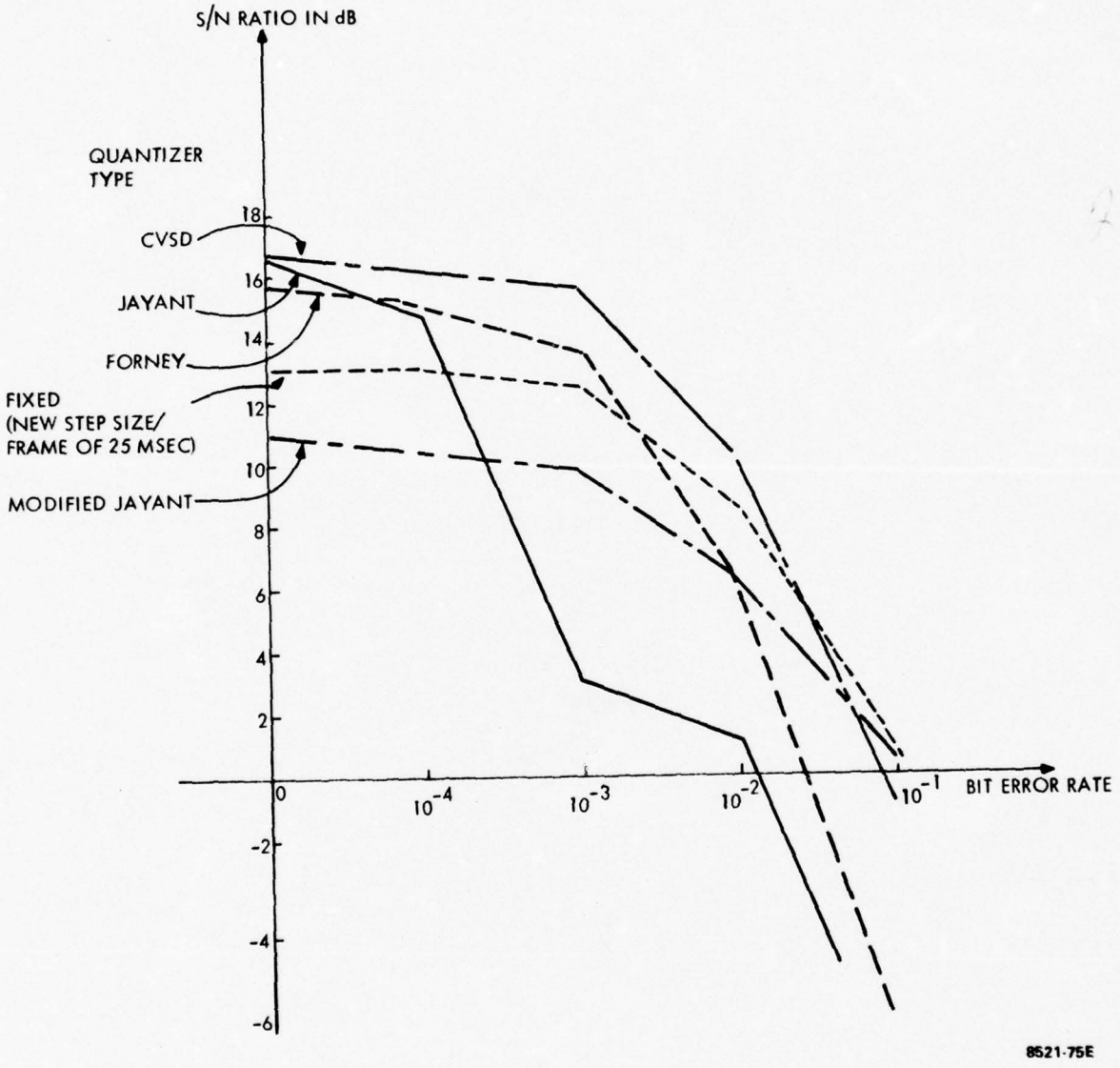


Figure 2-12. S/N of Different Quantizers versus Bit Error Rate (1st Order Adaptive Predictor W/O Pitch Loop)

As can be seen from Figure 2-12, the Jayant quantizer, CVSD and the Forney algorithms are within 1dB of each other in the absence of channel errors. Under this condition, the fixed/frame quantizer is 3dB below the performance of these other quantizers and the modified Jayant quantizer is 1dB below the fixed/frame quantizer. The Jayant algorithm, however, is extremely sensitive to channel errors. Its performance falls off rapidly when errors occur. This quantizer produces noticeable distortion at error rates as low as 10^{-4} and is unacceptable at BER's of 10^{-3} and above. The Forney quantizer is more robust, maintaining good performance at 10^{-3}, fair performance at 10^{-2} and unacceptable performance at 10^{-1}. While starting at a lower performance level than the other quantizers, the fixed frame quantizer does not degrade rapidly with increasing channel errors. In fact, it provides better performance than either the Jayant or Forney quantizers at error rates higher than 10^{-2}. The modified Jayant quantizer using a decay factor of 0.96 had a S/N that was always below the fixed frame quantizer, but at error rates above 10^{-2}, was higher than the FORNEY quantizer. Note that CVSD, the simplest system, exhibited the best performance against channel errors until error rates exceed 10^{-1}.

Based on the simulation data of Figure 2-12, the fixed frame quantizer was chosen to evaluate the performance of the APCQ system with more complex predictors. Figure 2-13 shows how APCQ systems using a fixed frame quantizer perform against channel errors over the same sentence material as Figure 2-12. In this figure, the channel errors are imposed uniformly on the transmitted data including the pitch M, pitch gain α, the PARCOR coefficients and the quantizer step size as well as on the quantized error signal sequence.

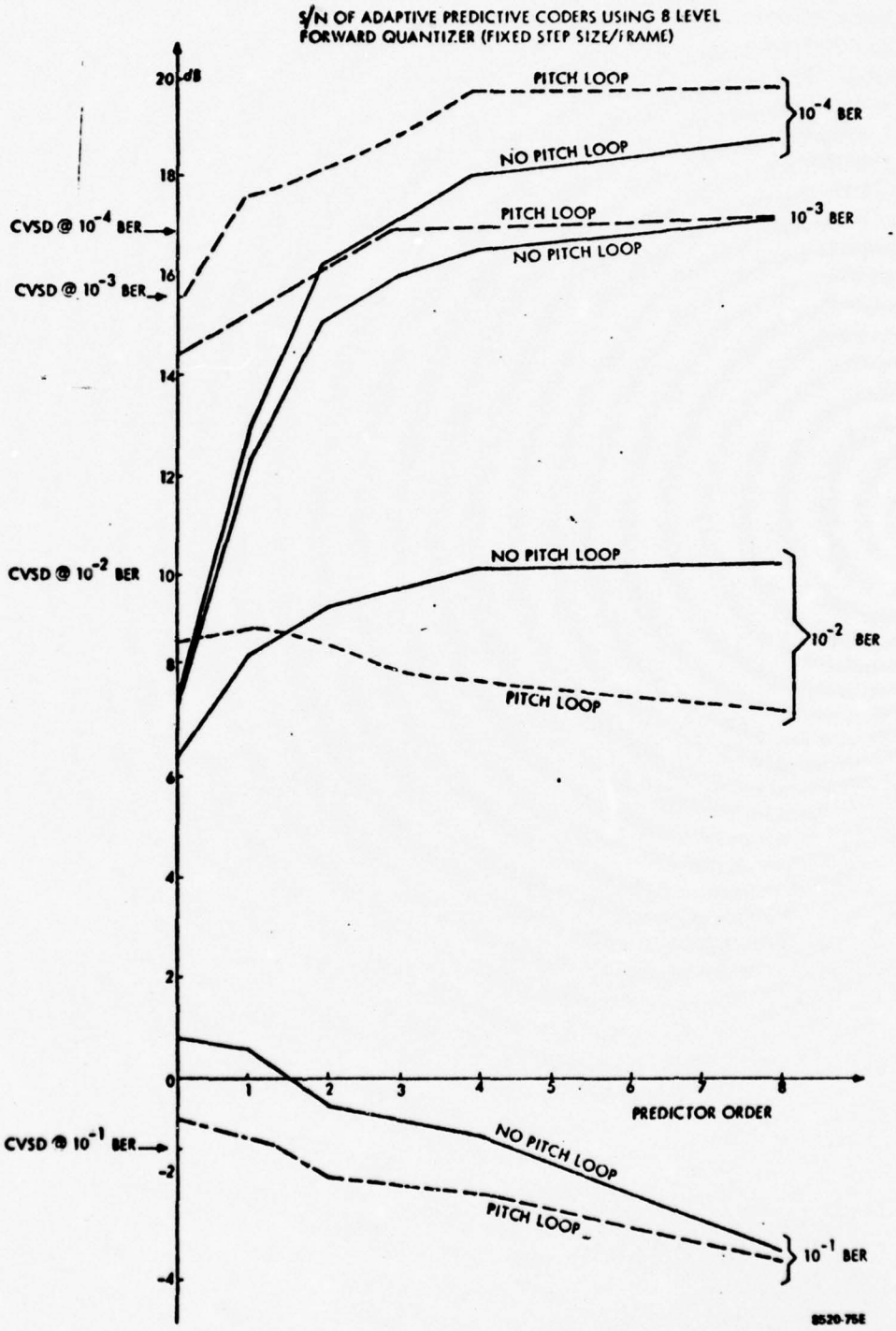


Figure 2-13. 16 KBPS Speech Coder Performance vs. Random Channel Bit Errors (BER)

At low error rates, a fourth order APCQ system with pitch outperforms CVSD by 3dB. As bit errors are imposed, the APCQ system loses its advantage over CVSD. At high error rates, the presence of the pitch loop becomes detrimental to APCQ performance. At error rates between 10^{-3} and 10^{-2}, the APCQ system with fixed/frame quantizer and CVSD have similar S/N rates. Note that the simplest system (i.e., APCM) outperforms CVSD at error rates of 10^{-1}. This is shown by the bottom curves of Figure 2-13 with predictor order equal to zero and no pitch loop present. Figure 2-14 provides an alternate method of graphing some of the data given in Figure 2-13 . These curves compare the channel error performance of a fourth order adaptive predictor, with and without pitch, to CVSD. At low error rates, fourth order systems are superior to CVSD. At higher error rates, CVSD is either superior to APCQ with pitch or comparable to APCQ without pitch.

These results indicate that system performance is a critical function of channel error rate. As low errors occur, the complex APCQ systems outperform CVSD at 16 Kb/sec. For high channel error environments, CVSD is superior. If high quality is the objective in low error environments, then the APCQ system provides the designer with a superior voice digitization technique.

Based on these simulation results, GTE Sylvania and DCA agreed to develop a final system using a fixed frame quantizer. To increase the audio bandwidth from 2500 Hz to 3200 Hz, the 8 level quantizer was reduced to a 5 level quantizer. This lowered the S/N ratio of the APCQ system approximately 3dB from those presented in Figures 2-11 through 2-14.

2.4.3 Improved Error Signal Quantizers

The modified Jayant algorithm, however, did not become available until after we made this decision to implement with the

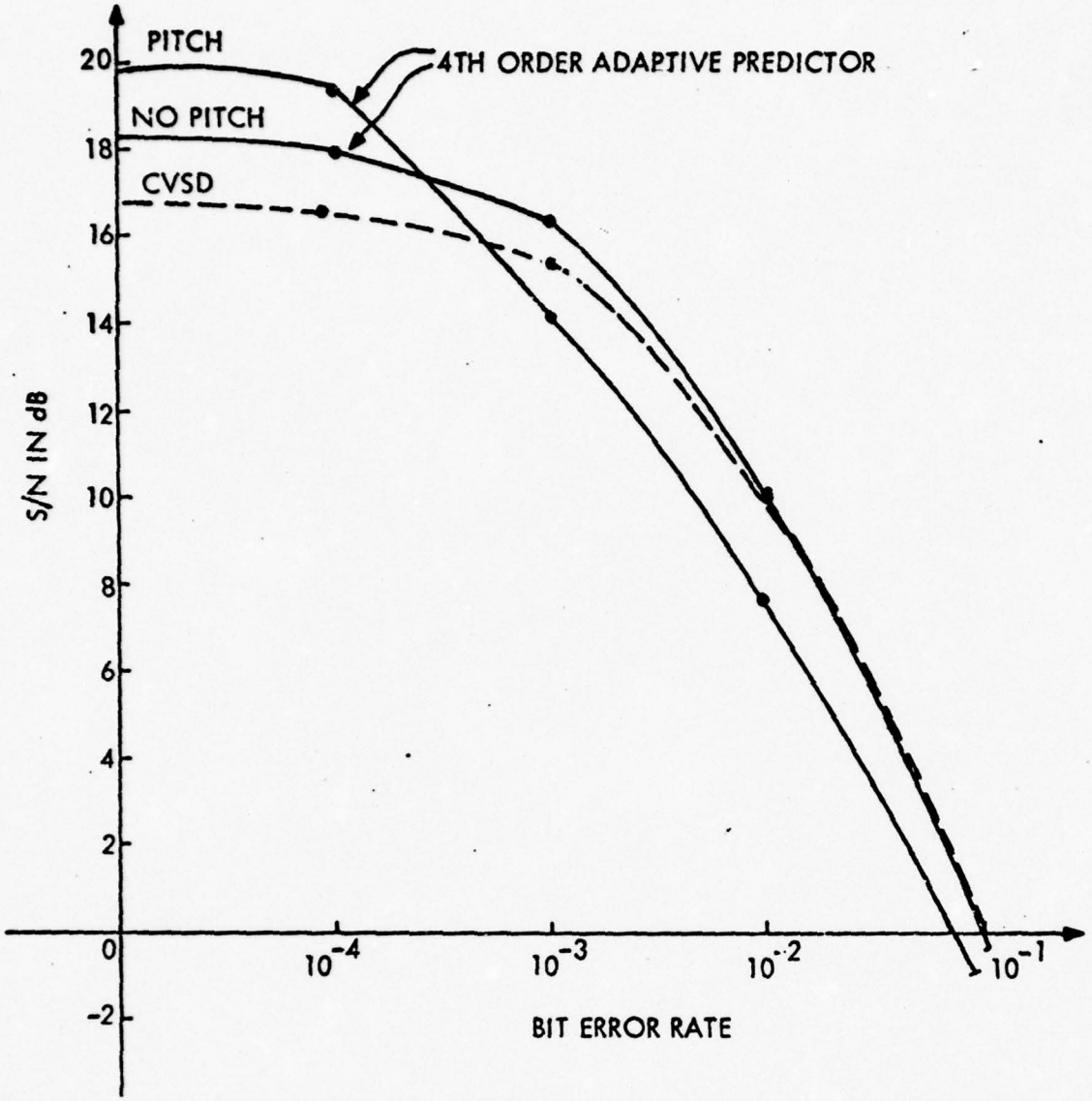


Figure 2-14. S/N of CVSD and 4th Order Adaptive Predictive Coder (Fixed Quantizer) vs. Bit Error Rate @ 16 KBPS

8523-75E

fixed/frame quantizer. Moreover as Figure 2-12 illustrates, its S/N is always lower than the other quantizers. Nevertheless, informal listening tests conducted after the decision to implement with the fixed/frame quantizer indicated little if any degradation between the Jayant and modified Jayant quantizers without channel errors.

Even more important was the discovery that the distortions introduced by the modified Jayant quantizer were less objectionable than those introduced by the fixed/frame quantizer. Consequently, during the last weeks of the contract, additional simulation runs were performed to observe the behavior of the modified Jayant algorithm further.

Figure 2-15&16 illustrates the performance of the APCQ coder with channel errors inserted into the transmitted data using the modified Jayant quantizer. The S/N ratios behave about the same way as the fixed frame quantizer in the presence of channel errors. Thus for error exponents greater than 10^{-3}, the systems having no pitch predictor and a low order coefficient predictor performed better than the systems with a pitch loop and high order predictions. The simple systems are better since the channel errors occur on the pitch parameters and predictor coefficients and these must significantly affect the S/N ratio. On good channels (BER $\leq 10^{-4}$) the S/N ratio improves as the predictor becomes more sophisticated.

The S/N ratio of the APCQ coder with modified quantizer is always lower than that for the fixed/frame quantizer. The perceived voice quality, however, is generally higher because the types of distortion are different.

Thus the S/N curves while indicative of the expected performance of a technique are not always directly related to voice quality.

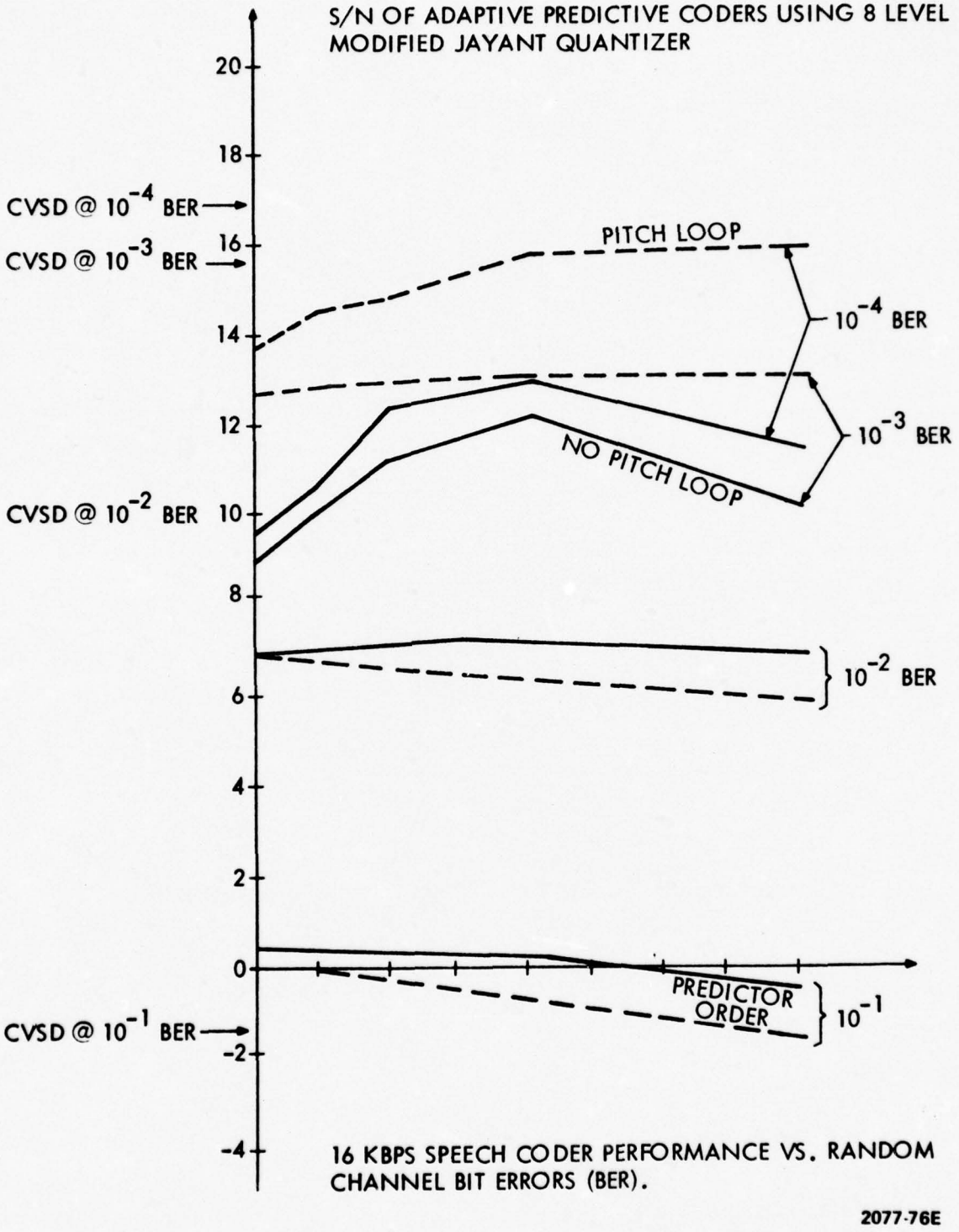


Figure 2-15. S/N of Adaptive Predictive Coders Using 8-Level Modified Jayant Quantizer

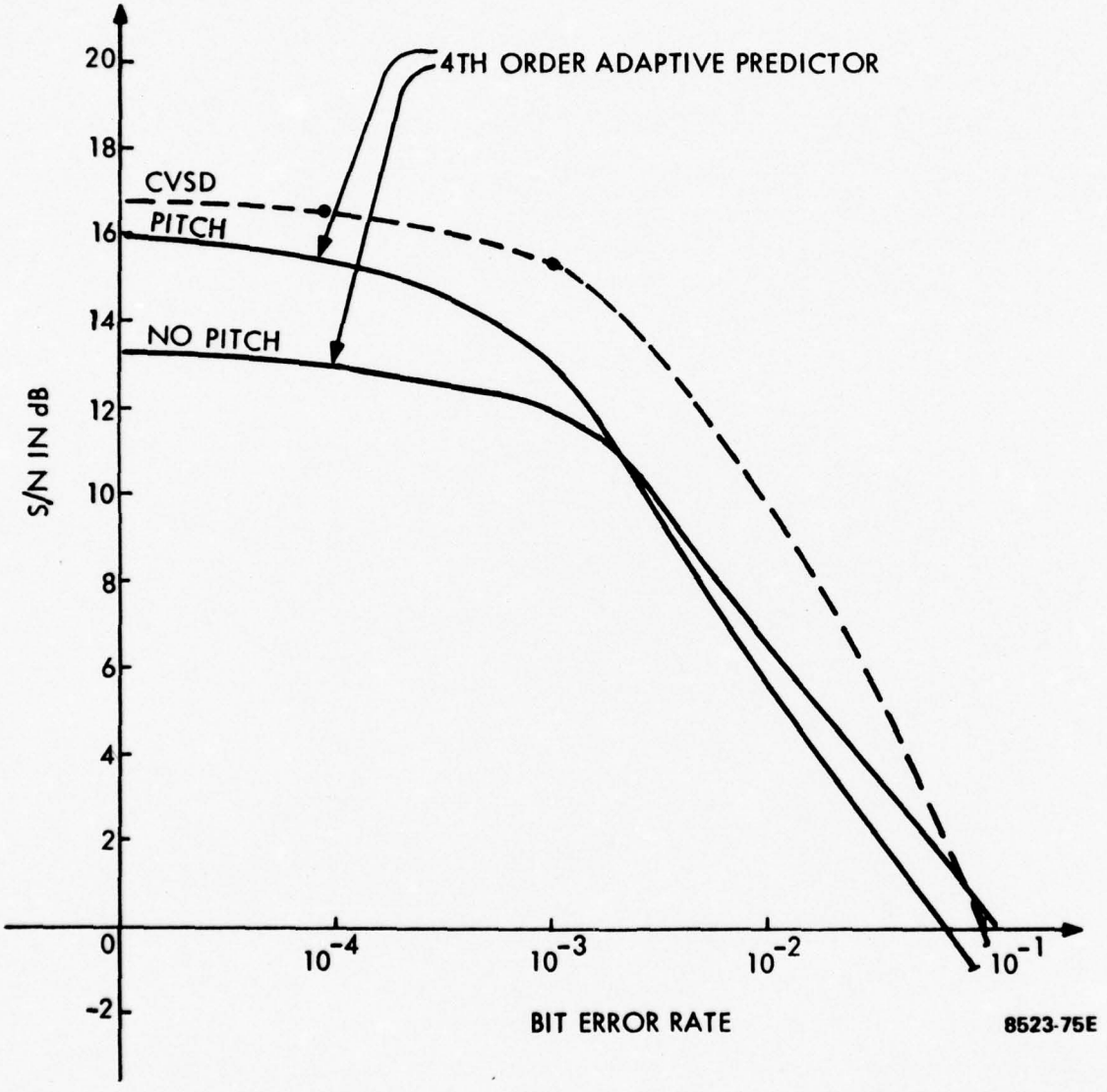


Figure 2-16. S/N of CVSD and 4th Order Adaptive Predictive Coder (Modified Jayant Quantizer) versus Bit Error Rate at 16 Kb/s

The modified Jayant quantizer has high voice quality but fairly low S/N ratios.

The modified Jayant quantizer because it does not degrade as the predictor becomes less complex offers the potential of simplifying the APCQ system dramatically.

SECTION 3
Real-Time Software

3.1 Description of APCQ Program

3.1.1 General Discussion

The real-time APCQ program written in assembly language enables the EDM to collect, analyse, transmit, receive and syntheize speech data in full duplex mode. The sequence of operations involves initialization of the system parameters, set-up of the interrupt processing, acquisition of synchronization, processing of speech via the APCQ algorithm, transmission and reception of parameters and reconstruction of speech.

The APCQ algorithm implemented is a fourth order adaptive predictive coder with a pitch loop and a five-level fixed frame quantizer. Each speech sample is predicted as a weighted linear combination of four past speech samples. The error signal, computed from the difference of the actual and the estimated speech sample, is then quantized into five levels by the quantizer shown in Figure 3-1 with level estimated from the RMS power of the error signal which remains constant throughout a whole frame of speech data.

The key to correctly sequencing the operations is the use of two interrupts in the EDM: the speech side interrupt and the line side interrupt. The speech side interrupt is set to interrupt the computer at regular intervals and transfer control to software (starting at Location 0), which inputs one speech sample into the computer and outputs one speech sample from the computer. Careful count is kept of the number of samples read in; and when a frame of data has been input, an indicator (transmit flag) is set showing the analysis can begin.

The line side interrupt also regularly interrupts the computer and transfers control to software (starting at Location 1), which outputs modem clock (five-volt levels from a flip-flop) and one bit of transmission data. At every eighth interrupt it inputs eight bits of

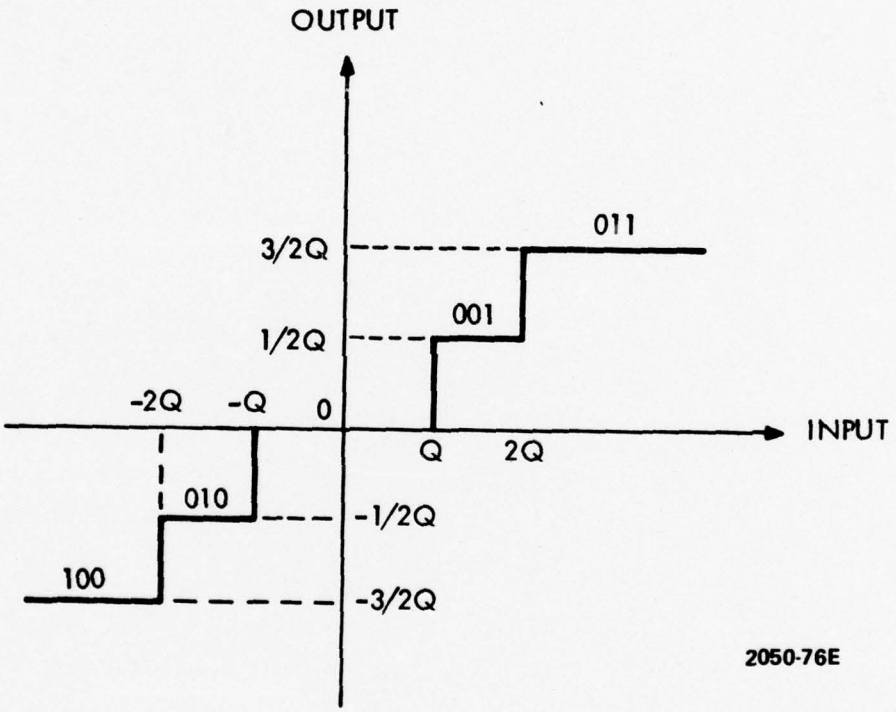


Figure 3-1. Characteristics of a 5-Level Quantizer

received data which have been accumulating in an eight-bit shift register. When it receives a frame of input data, it sets an indicator (receiver ready flag) showing that synthesis can begin.

Since the program operates in less than real-time, it executes instructions in a wait loop while it idles. It leaves the wait loop to analyze speech whenever it sees that the transmit ready flag has been set by the speech interrupt. After analysis it returns to the wait loop where it will leave again for the synthesis when the line side interrupt raises the receive ready flag. In addition, it also interrupts the sync flag and leaves for the sync search routine whenever the flag is set. Naturally, the speech and line interrupts are functioning even during execution of the wait loop. Speech is input and output at each sampling instant, and data is sent and received at 16,000 bps.

3.1.2 Initialization and Wait Loop

When the program is started, it immediately goes to an initialization mode where all parameters and counters of analyser and synthesizer filters are cleared. It then goes into the wait loop shown in flow chart Figure 3-2 and waits until one of three flags, namely: transmitter ready flag, receiver ready flag and sync search ready flag is set where it performs the corresponding processing. At the end of the operation, the program returns to the wait loop to use up the idle time left over. Reinitialization can always be achieved by setting one of the front panel switches.

3.1.3 Synchronization

After received 80 bauds (1.5 second) of data, the sync search ready flag is set where the program goes to sync search loop shown in Figure 3-3.

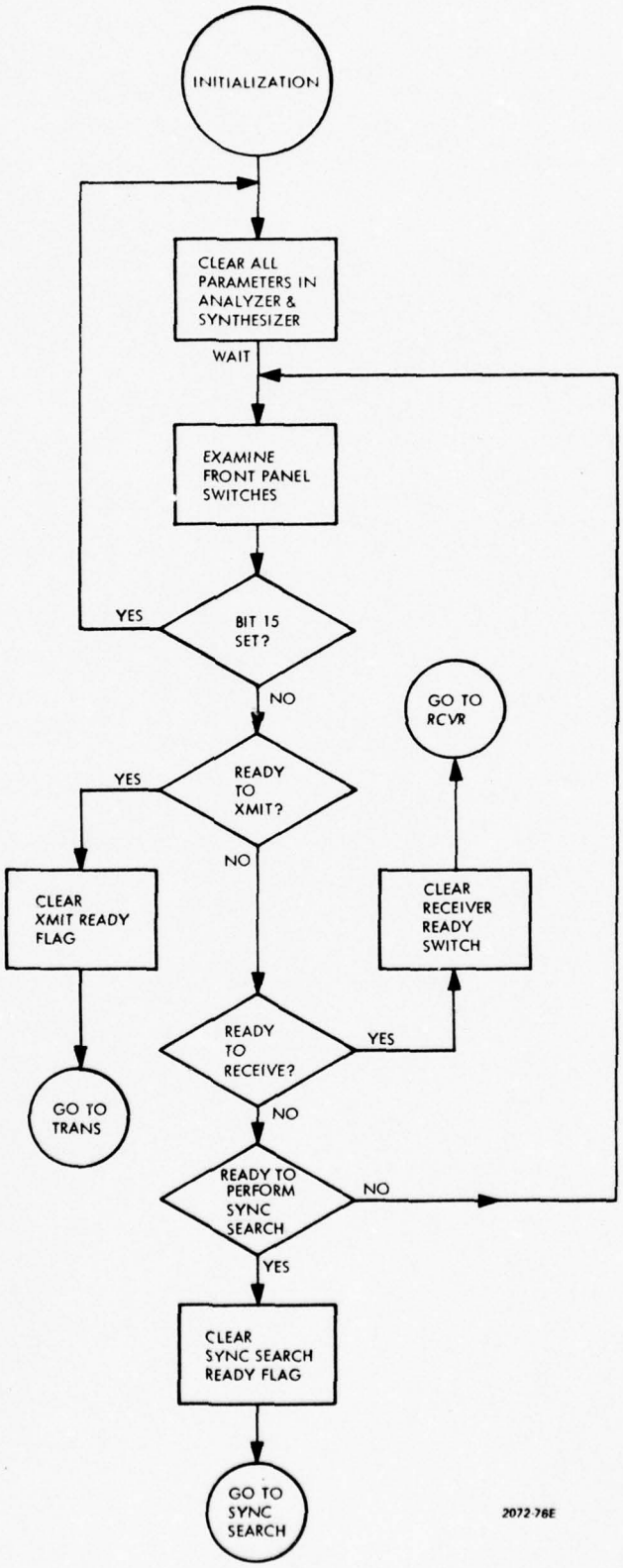


Figure 3-2. Initialization and Wait Loop

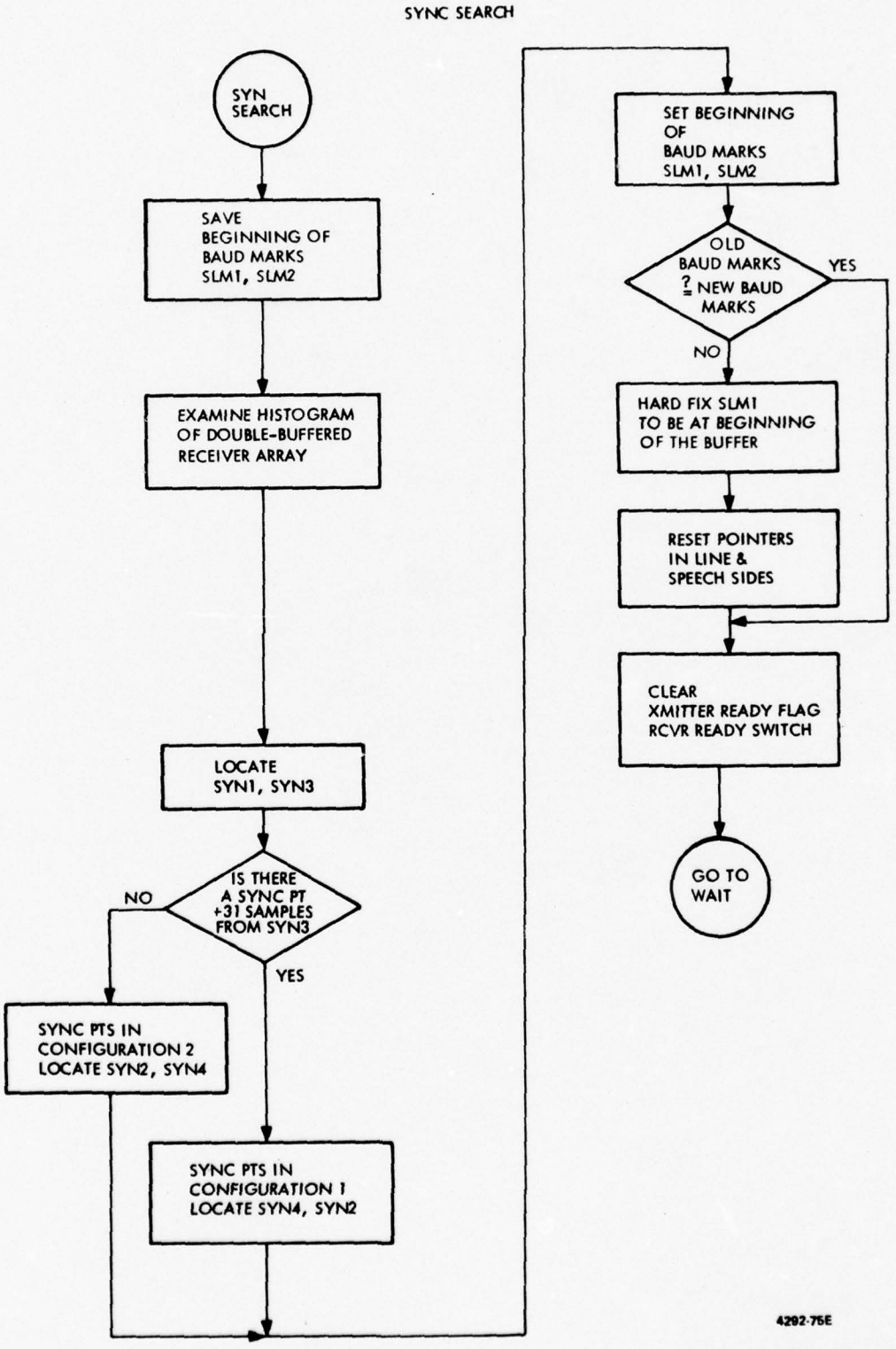


Figure 3-3. Synchronization Search

To understand the synchronization process, it is necessary to describe the format of the transmission data. Figure 3-4 shows that the transmission baud consists of 312 bits arranged in the following manner: 280 bits for the predictor error signal, one synchronization bit which is always set to a one, three bits for fourth PARCOR coefficient, three bits for the third PARCOR coefficient, four bits for the second PARCOR coefficient, four bits for the first PARCOR coefficient, three bits for the pitch gain ALPHA, six bits for the pitch, four bits for the RMS power of error signal, three bits for the normalizer and a second one bit for synchronization which is also always a one. The receiver develops a histogram of the received data, counting the times a one is received in each of the 312 bit positions. After 80 frames (approximately 1.5 seconds) the receiver examines the histogram for positions of two values near 80 (the values may be slightly less than 80 because of transmission errors) which are 31 bit positions apart. From the position of the sync points, the software then knows where the last bit in the frame is located, and it shifts the received sequence so that the first bit of the baud lines up with the first storage location. Once synchronization has been achieved, the locations of the synchronization bits are saved and compared with that of subsequent sync searches to determine the necessity of shifts in the following received sequence.

3.1.4 Line-side Interrupt

Figure 3-5 describes the program flow whenever the clock interrupts the computer for data transmission or reception by trapping to Location 1. First the flip-flop which drives the modem clock is set

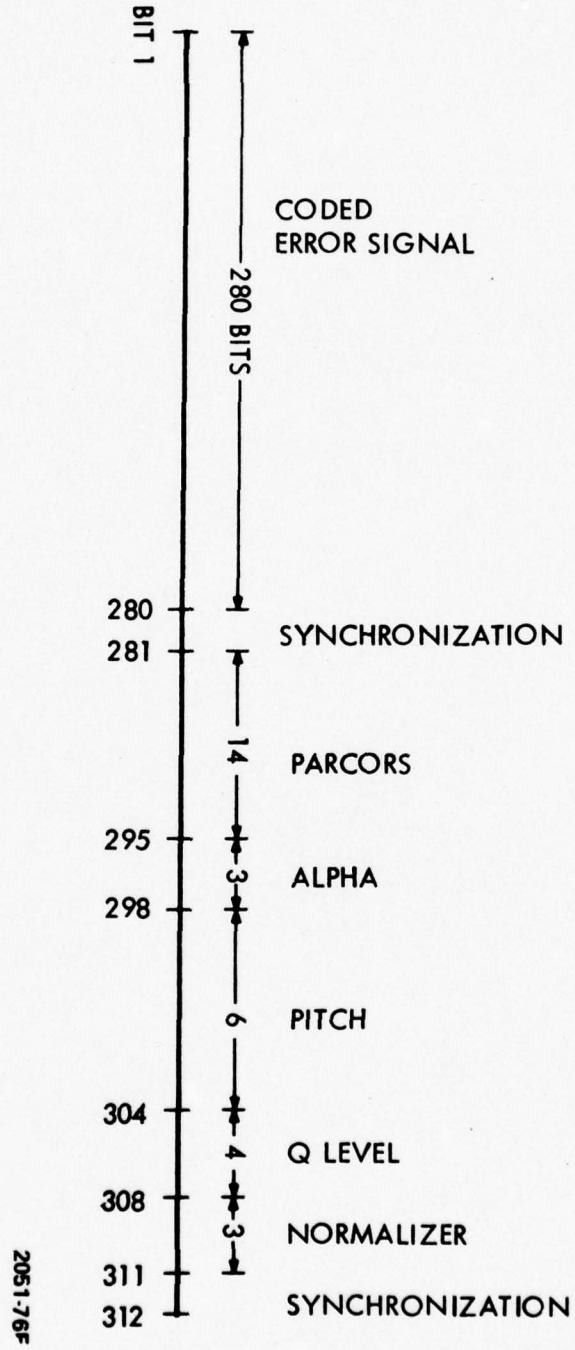


Figure 3-4. Format of Transmission Data

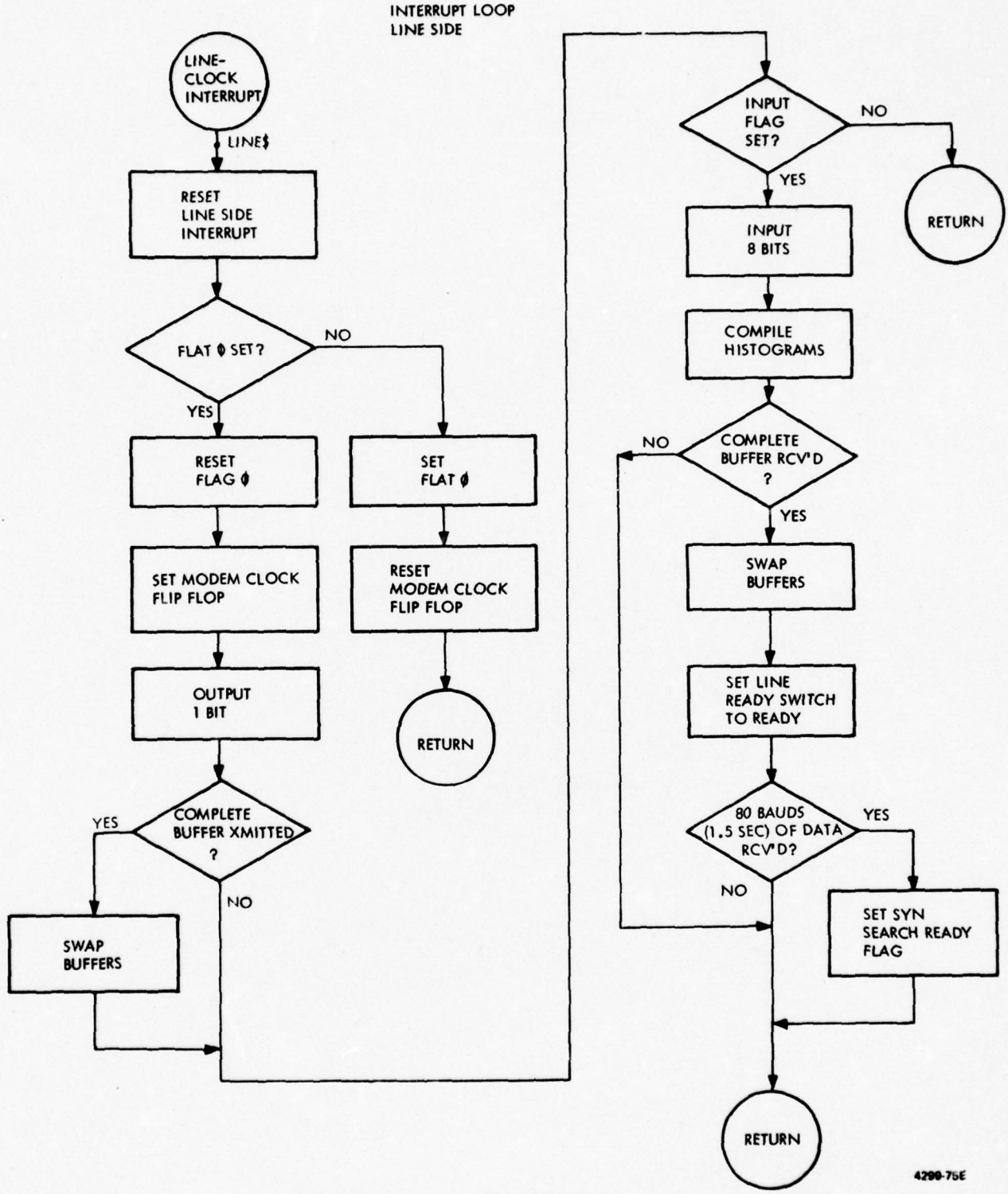


Figure 3-5. Line Side Interrupt

or reset depending on whether the interrupt was even-numbered or odd-numbered. Whenever it is set, one bit of data is also sent and the receiver buffer is interrogated to see if eight bits of data have been received. If received data is ready, the software inputs the data, compiles the histogram, and, whenever the baud is complete, sets a receiver ready flag. Every 80 bauds it sets a flag indicating that a synchronization search could begin if desired.

3.1.5 Speech Side Interrupt

The speech side interrupt is straightforward, as shown in Figure 3-6. When the computer interrupts and traps to Location 0, it reads in one speech sample, reads out one sample, and when the buffer is full, sets the transmit ready flag. The speech data is double-buffered to prevent loss of data during analysis.

3.1.6 Speech Analysis and Synthesis

Flow charts of Figures 3-7, 3-8, 3-9 and 3-10 describe the operations of the analysis of the APCQ coder. Upon entry to the analyser the input data is first normalized. Then pitch is computed via the AMDF function as shown in Figure 3-5. Next the pitch gain ALPHA and the reduced waveform are calculated as illustrated in Figure 3-7.

In Figure 3-8 the autocorrelation coefficients of the reduced waveform are determined that are later used to compute PARCOR coefficients via the matrix inversion routine shown in Figure 3-9 from which actual predictor coefficients (a's) are derived as in Figure 3-14. Finally, the RMS power of the error signal is calculated and used to quantize error signal and coded into seven bit words as shown in Figure 3-10.

Figure 3-12 and 3-13 describes the operations of the synthesizer where all transmitted parameters are decoded and speech is reconstructed by inverse filtering.

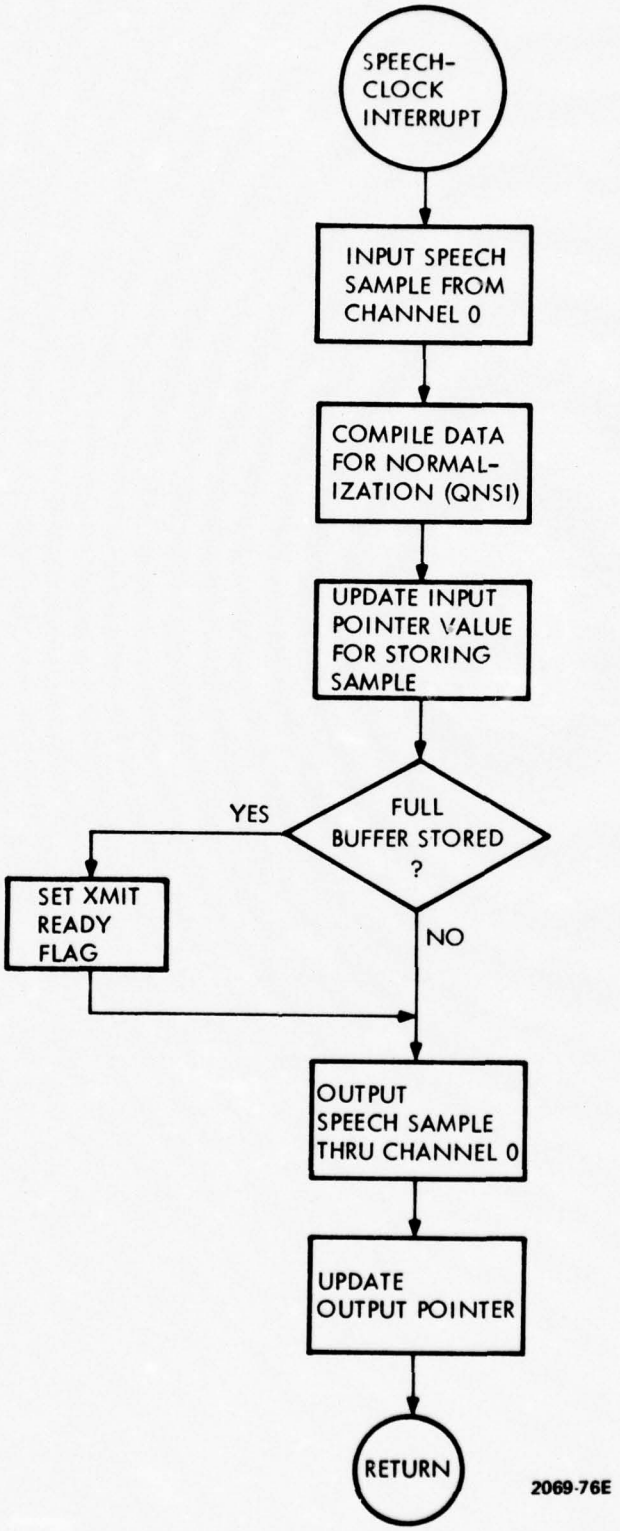


Figure 3-6. Interrupt Loop Speech Side

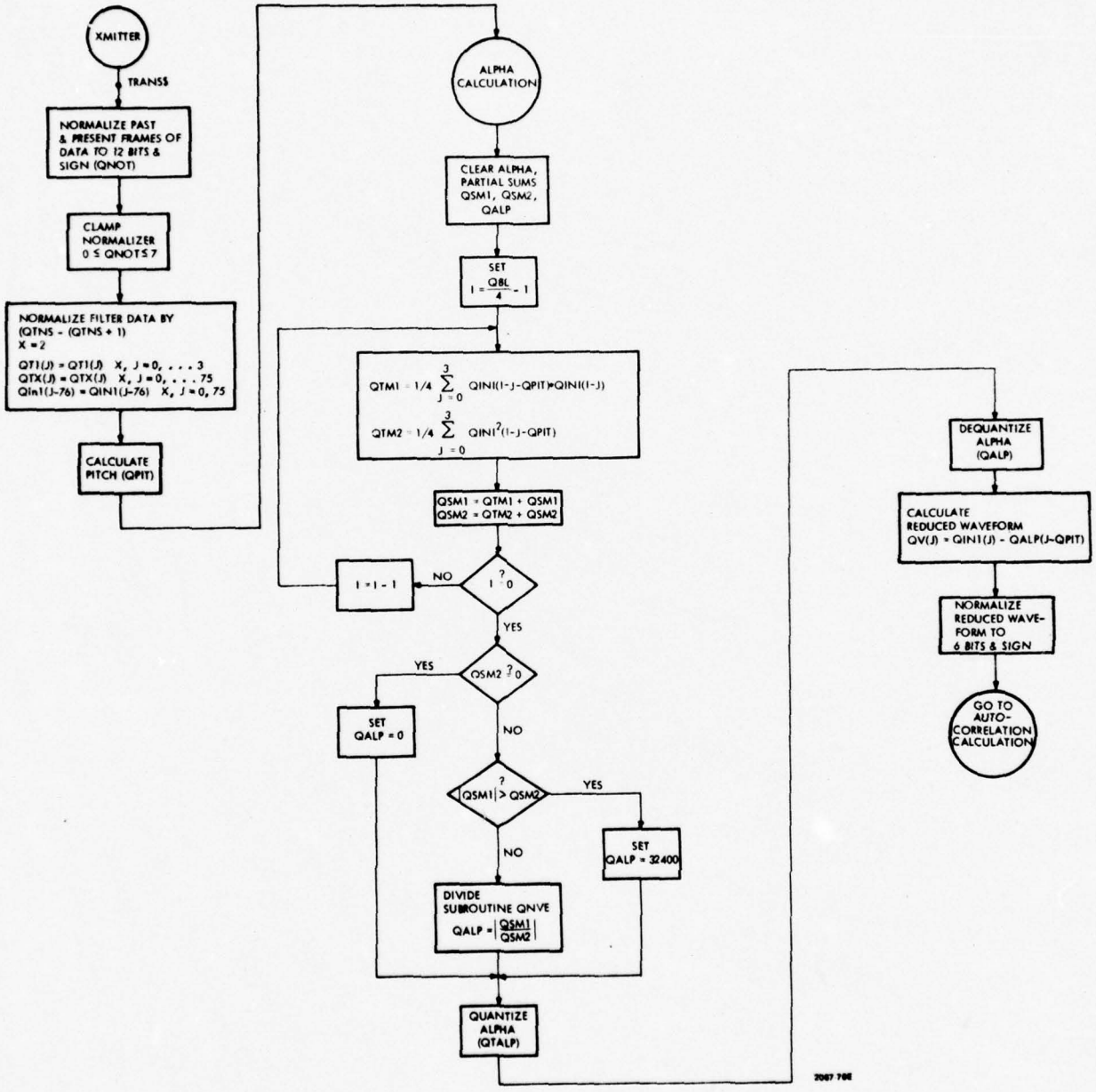


Figure 3-7. Analyzer for APCQ Coder

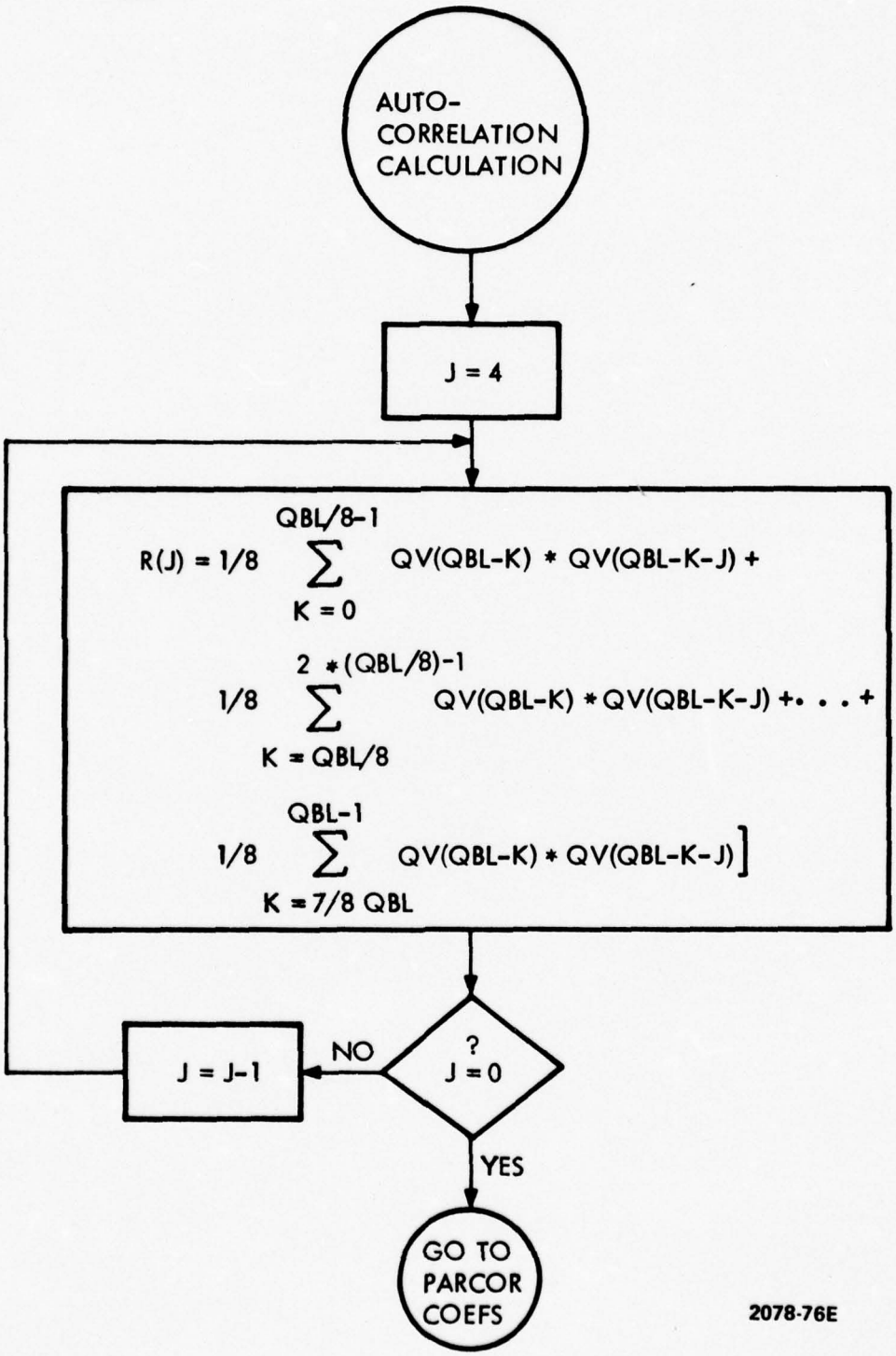


Figure 3-8. Autocorrelation Calculation

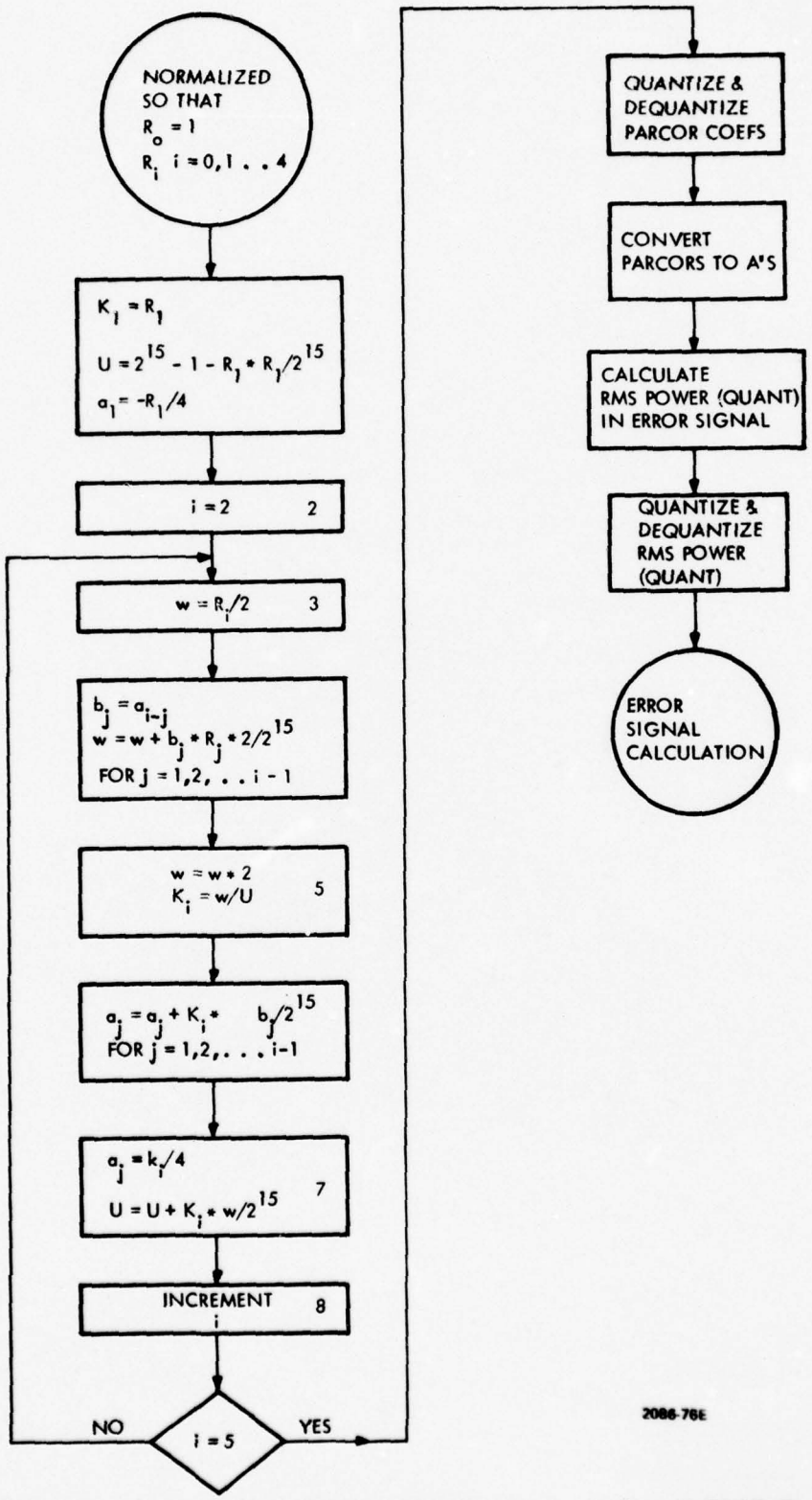


Figure 3-9. Calculation of PARCOR Coefficients (K_i) and RMS Power

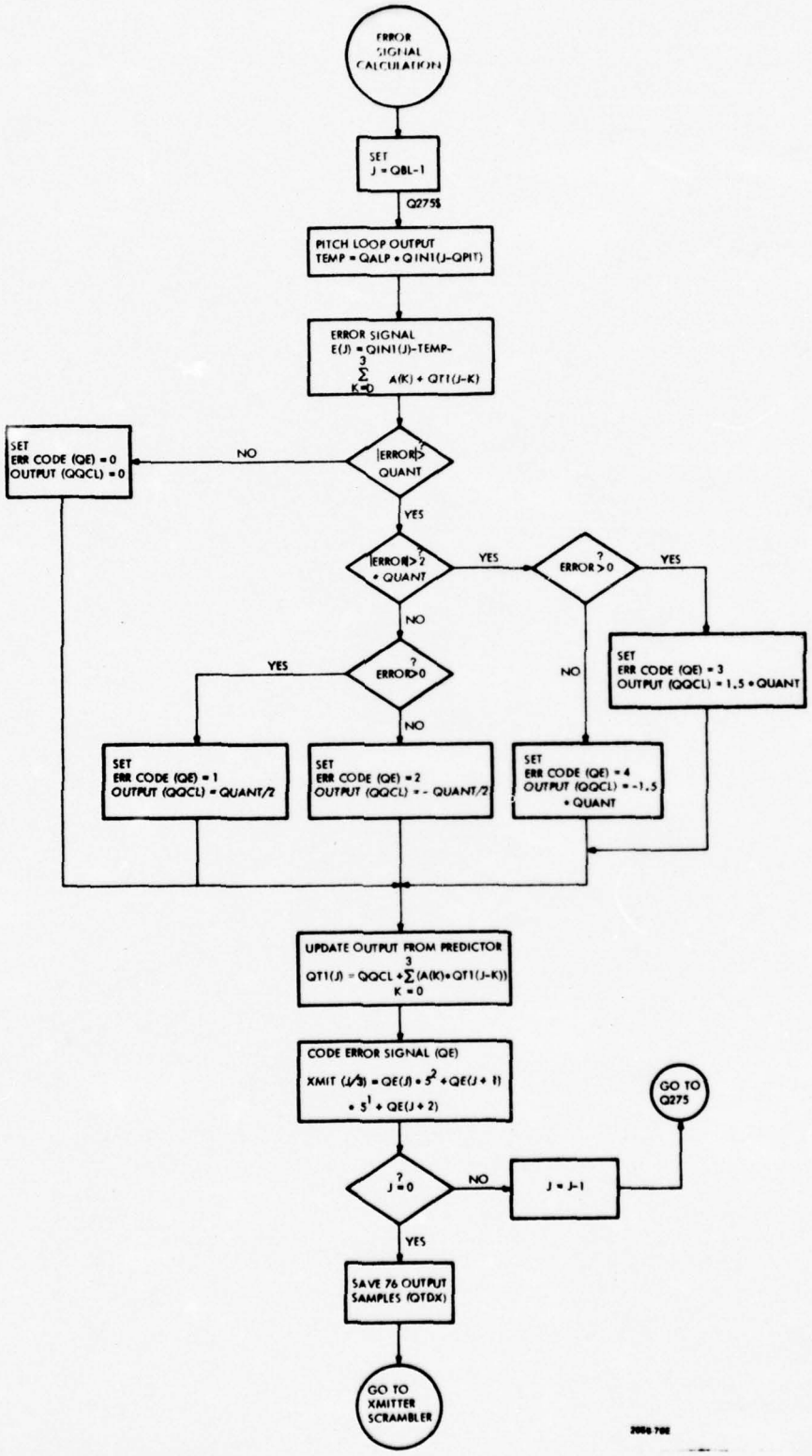


Figure 3-10. Error Signal Calculation and Quantization

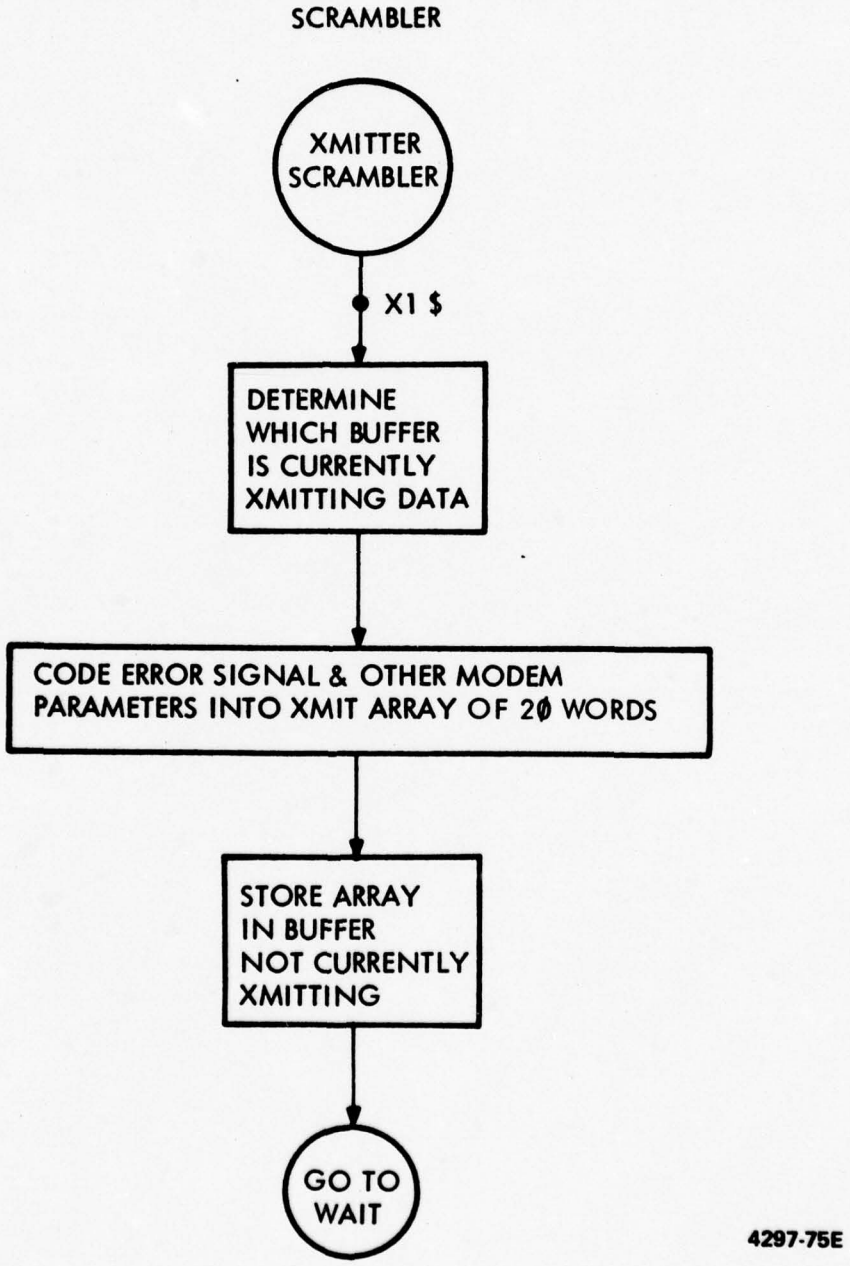


Figure 3-11. Transmission Data Construction

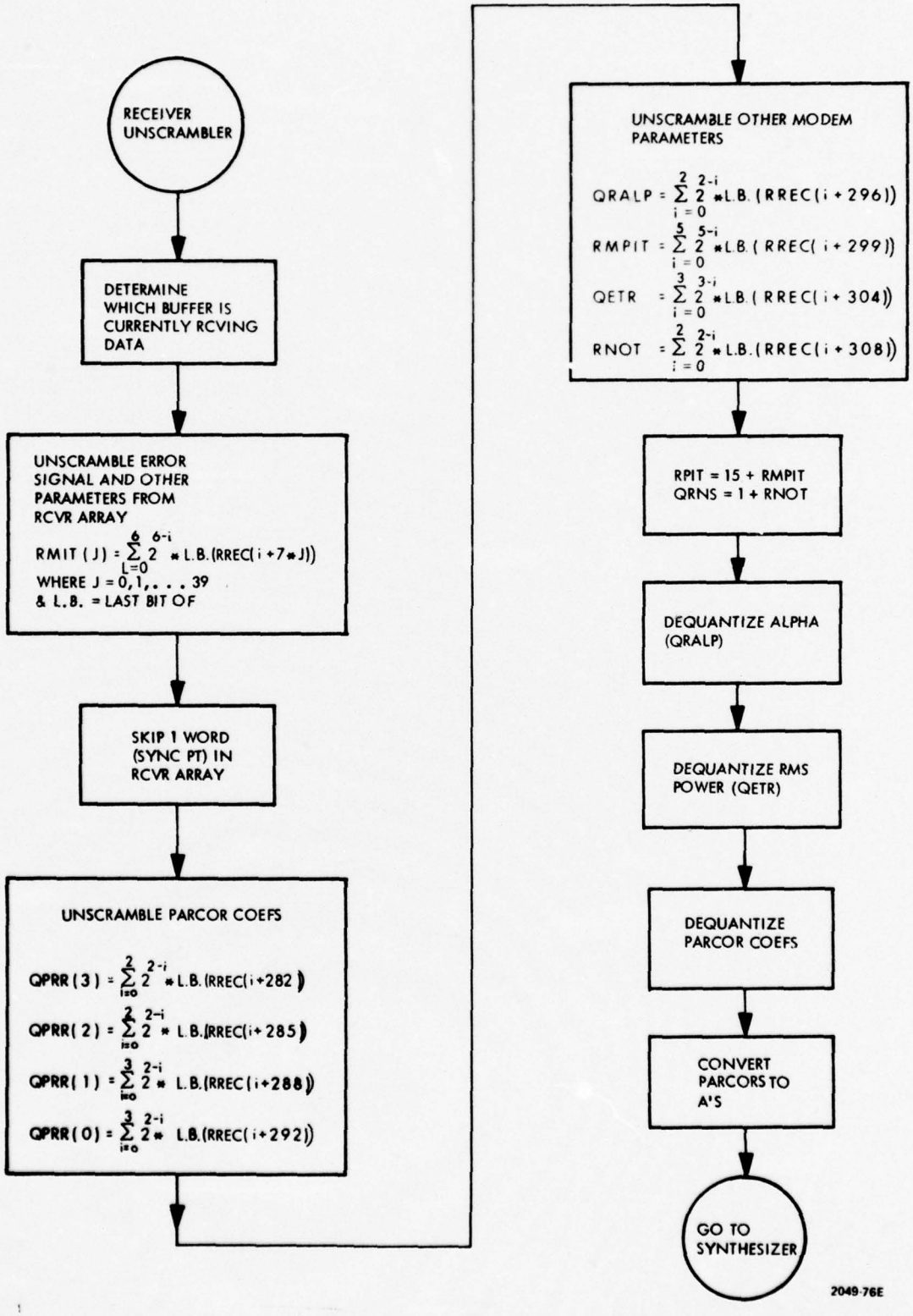


Figure 3-12. Data Reception, Decoding and Dequantization

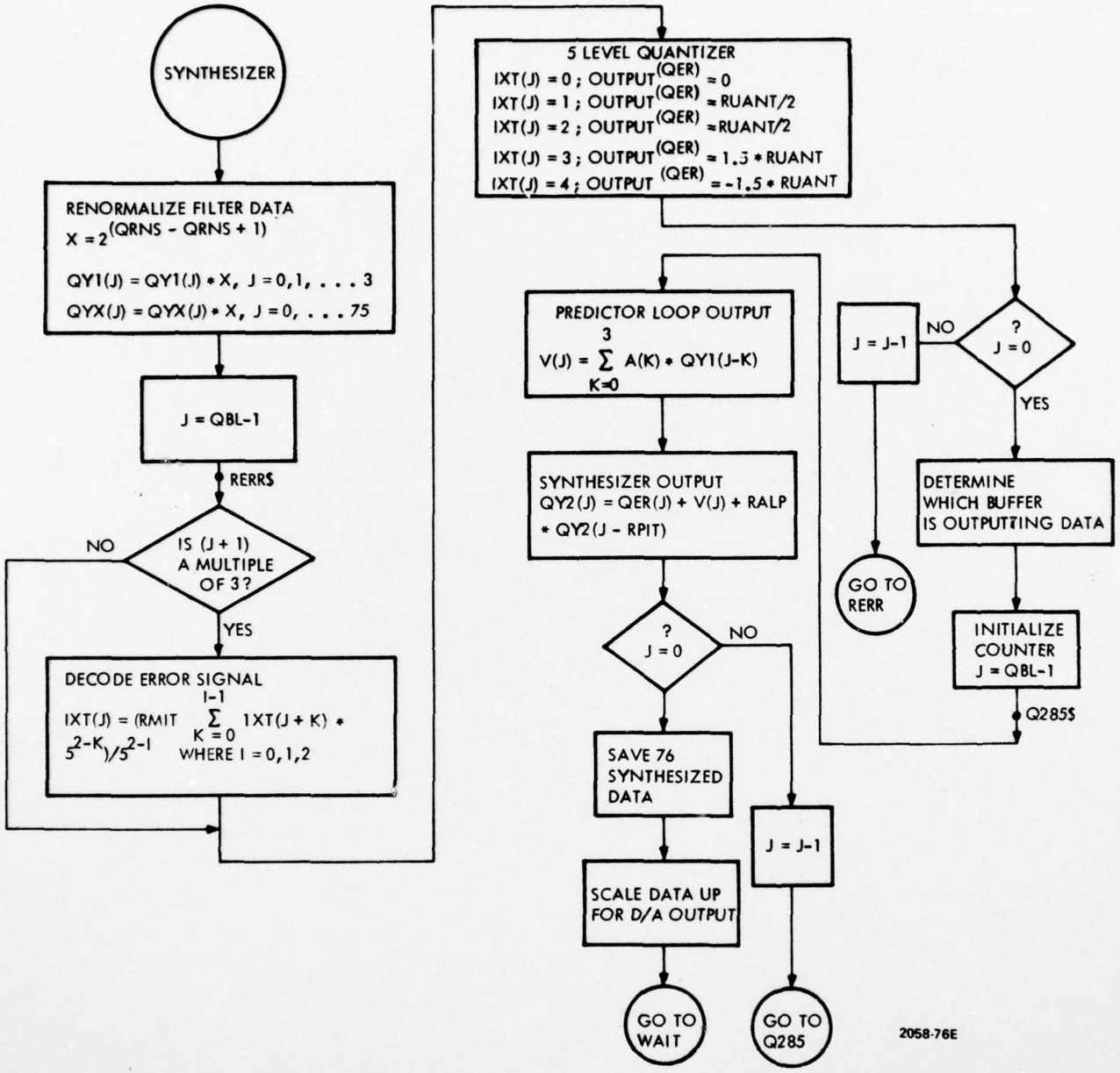


Figure 3-13. APCQ Synthesizer

SUBROUTINE QCAT & QACR

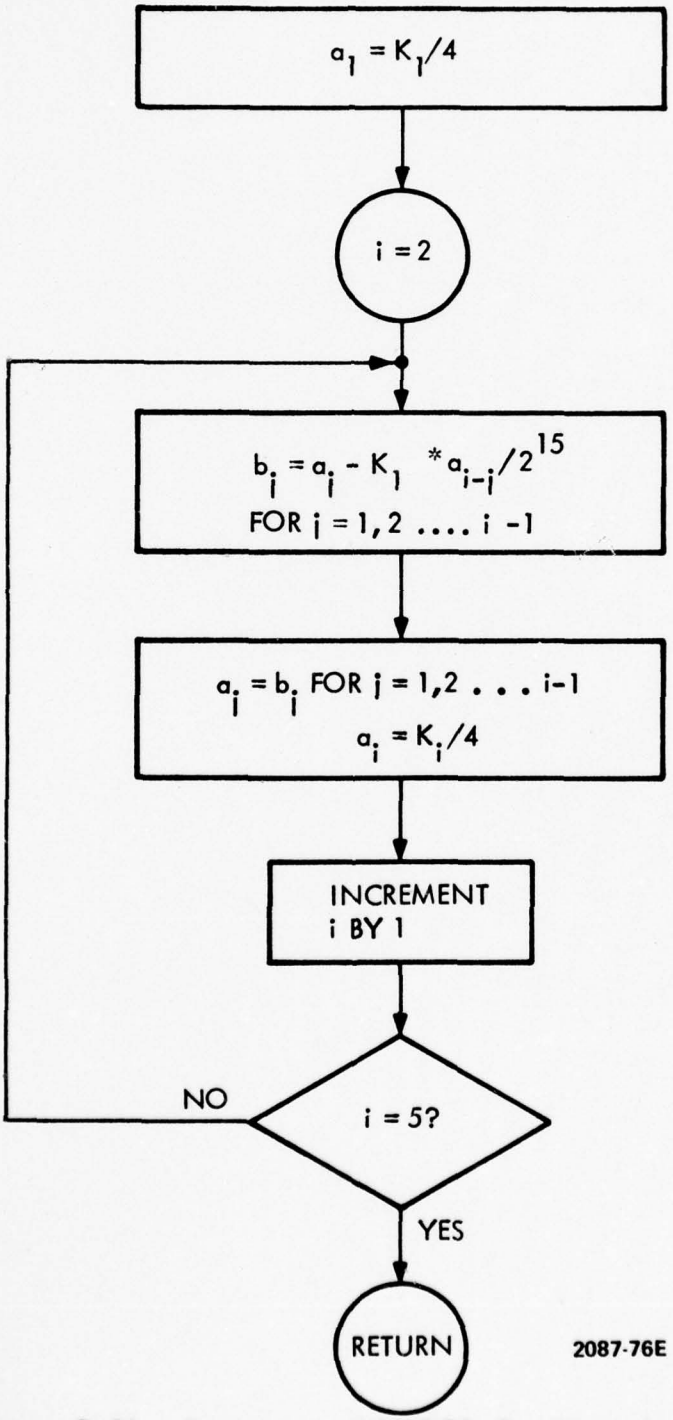


Figure 3-14. Converting PARCOR Coefficients to A's

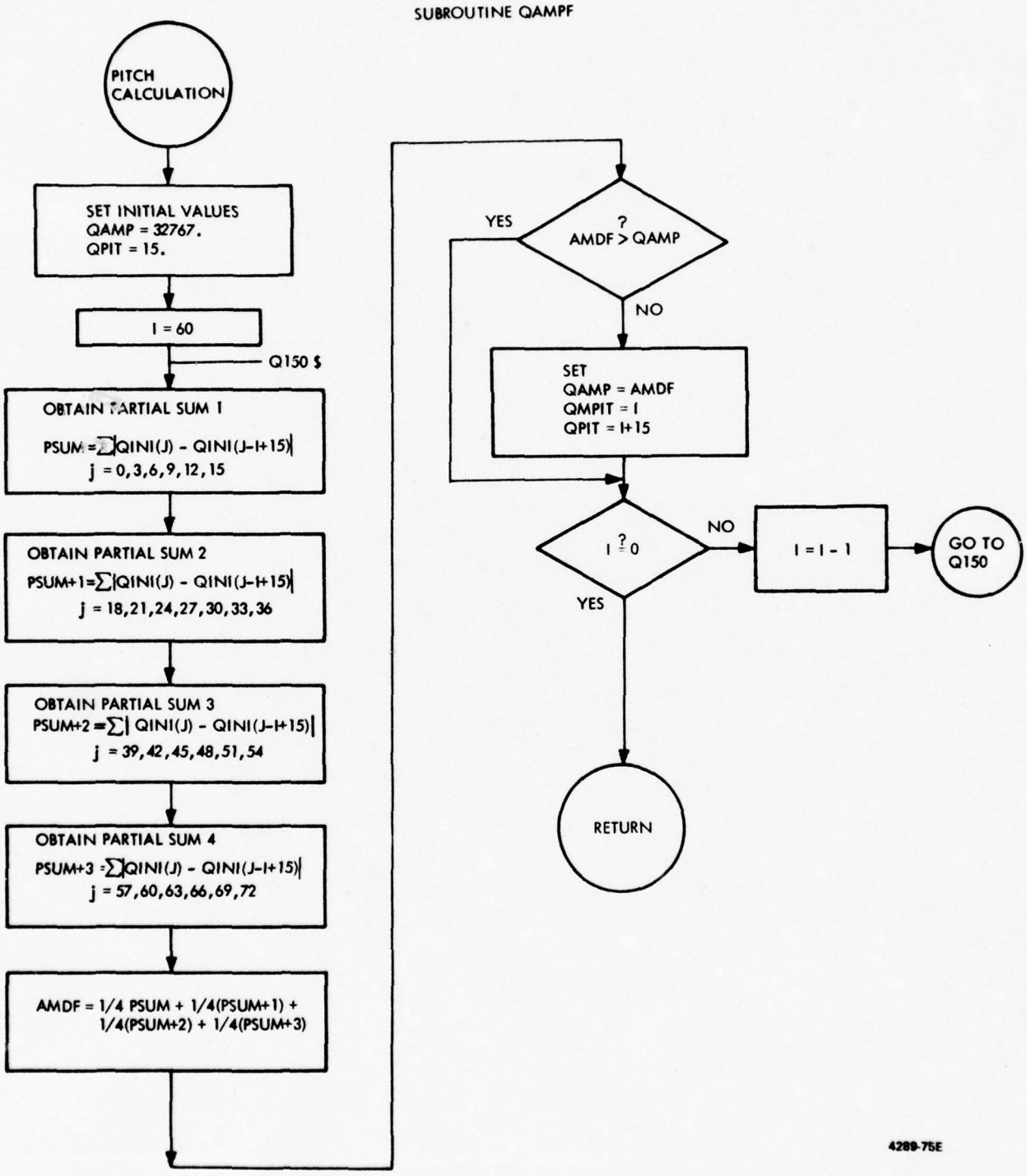


Figure 3-15. AMDF Pitch Calculation

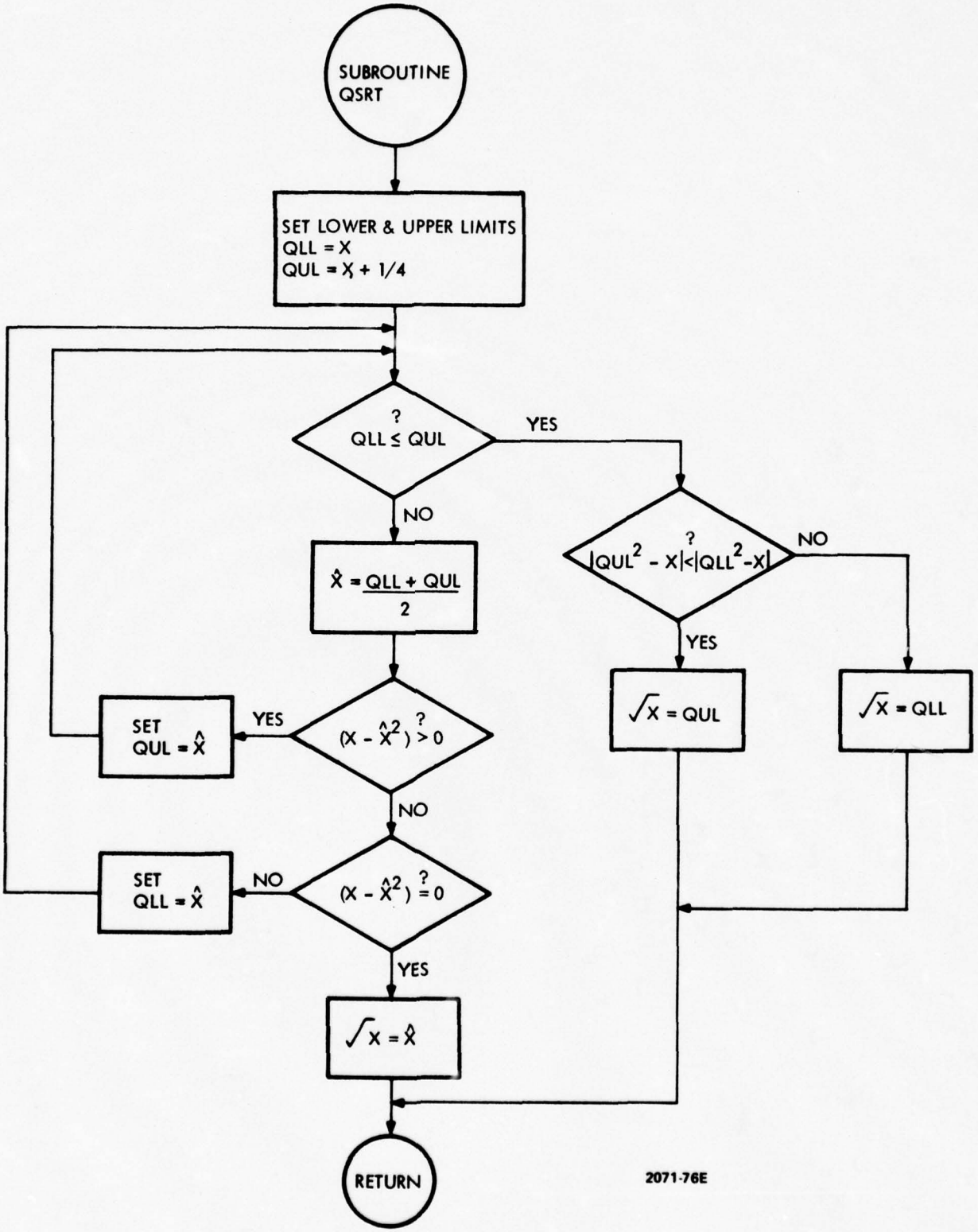


Figure 3-16. Subroutine QSRT

3.2 Operation Procedures for the APCQ Program

3.2.1 Connecting the System

a. Connect the EDM's to each other either in back-to-back mode shown in Figure 3-17 or through a 16,000 bps modem as shown in Figure 3-18.

1. Use the terminal marked "Data" on the rear of the CPU to connect to the modem or the other EDM.

2. Place the toggle switch to appropriate RS-232 or MIL-188 interface position as required

3. Strap the modem to operate from the external transmit clock or the leading edges of the data

b. Connect the handsets or audio tape recorders to the proper input and output jacks at rear of CPU

3.2.2 Reading in the EDM Software

a. Place all switch register toggle switches in the up position

b. Power on the PSP and card reader

c. Depress the "PM CLEAR" switch to clear program memory

d. Set "NORMAL/TEST PMIR" switch to "NORMAL"

e. Set "RUN/STOP" switch to "STOP"

f. Load the binary deck and depress the "MOTOR" and then "START" switches on the card reader

g. Cards (load PM, load DM, PMMA checksum, DM checksum) are read in. Card reader halts before it gets to PMIR checksum cards

h. Depress "STOP" switch on card reader

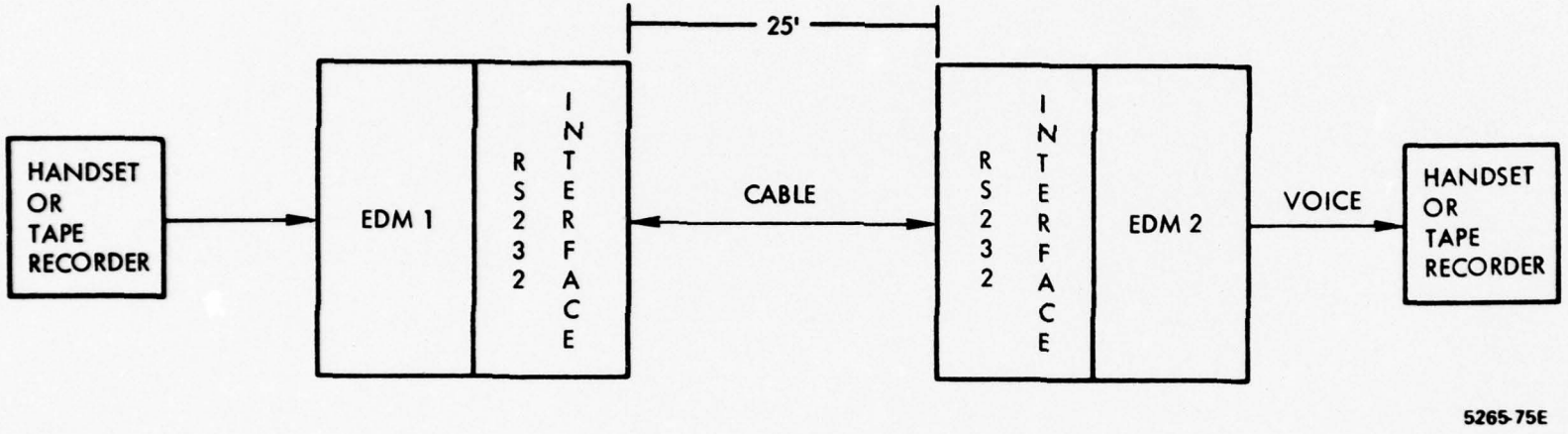


Figure 3-17. System Test Configuration

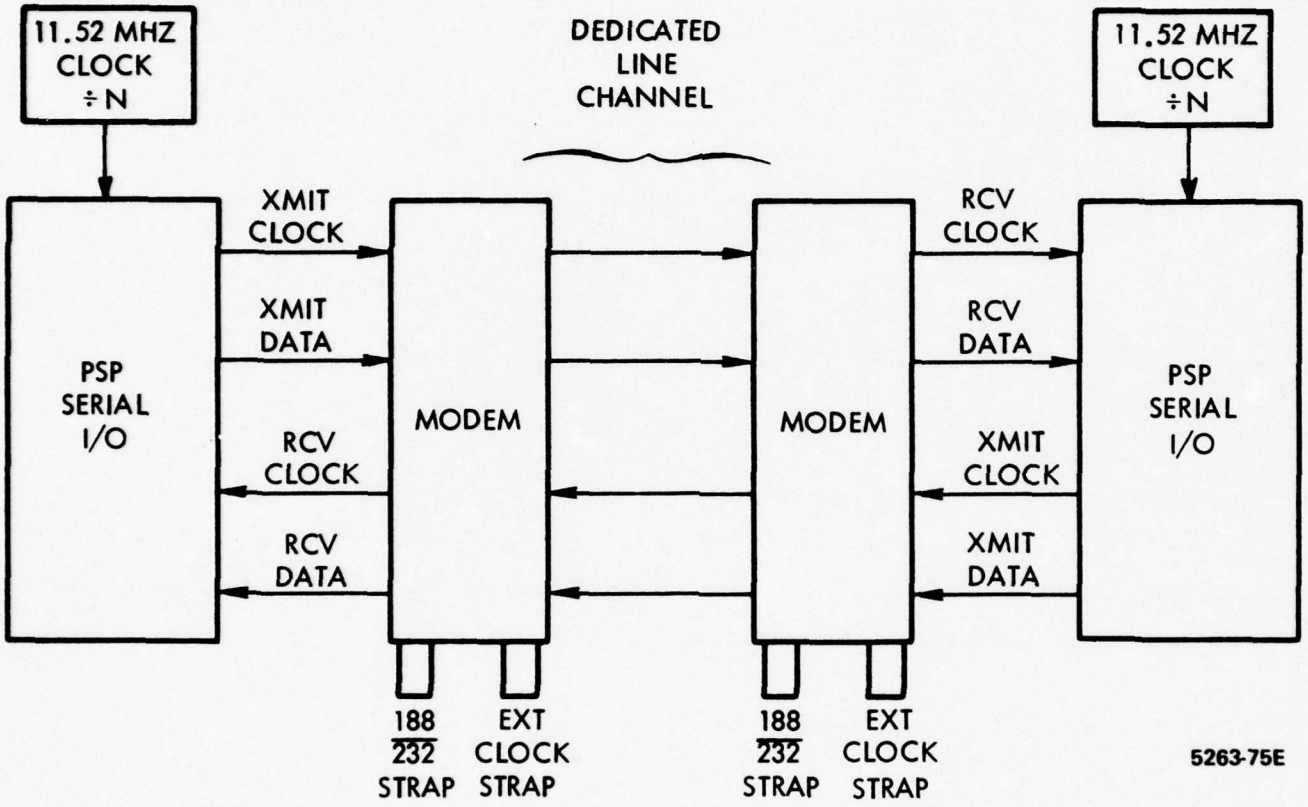


Figure 3-18. Modem Interface Signals and Interconnection

i. Set "NORMAL/TEST PMIR" switch to "TEST PMIR." Depress "MANUAL INST" switch

j. Restart the card reader by depressing "MOTOR" and "START" switches

k. After the whole deck is read, set "NORMAL/TEST PMIR" switch back to "NORMAL"

l. Set "INTERRUPT PROGRAM/DISABLE" switch to "INTERRUPT PROGRAM"

m. Set "RUN/STOP" switch to "RUN" to start executing the program.

3.2.3 Initializing the EDM

After the program has been running, reinitialization can be achieved by simultaneously holding switch register toggle switch 15 in the down position for each EDM and then returning it to the up position.

3.3 Description of Continuous Variable Slope Delta Modulation Program

3.3.1 Basic Principles of Operation

CVSD represents one of the simplest voice digitizers known, since there are relatively few mathematical operations per input sample. Figure 3-19 depicts a digital implementation of a CVSD transmitter and receiver operating on the DCA EDM's. The input speech is first converted to digital by a 12 bit A/D converter after lowpass filtering through an elliptic filter having a 3 dB frequency of 3200 Hz. The CVSD algorithm subtracts an estimate of the incoming sample from the input and quantizes the difference or error sample to +1 if the error is positive and -1 if negative. The quantized value is encoded for transmission by the mapping 1 into 1 and -1 into 0. The algorithm then uses a two state process to construct the estimate for the next incoming sample. The first stage computes a quantizer gain factor by interacting with time constant T_1 and uses this factor to adjust the level of the quantized error signal. The second stage filters this new error signal through an integrator having time constant T_2 to produce the new speech estimate. The CVSD receiver then reconstructs the ± 1 bit stream of the transmitter's error quantizer. This data excites the same quantizer gain and error signal integrator (which formed the speech estimate at the transmitter) and generates the synthesized speech after D/A conversion and low pass filtering.

With varying gain in the error signal, the system adapts to the slope of incoming speech waveform. Severe slope overload occurs with insufficient quantizer gain perceived as muffled speech with drop of volume. On the other hand, excessive gain will cause overshoots which introduce granular noise into the synthesized speech. In this study, three slope adaptation stretegies are investigated:

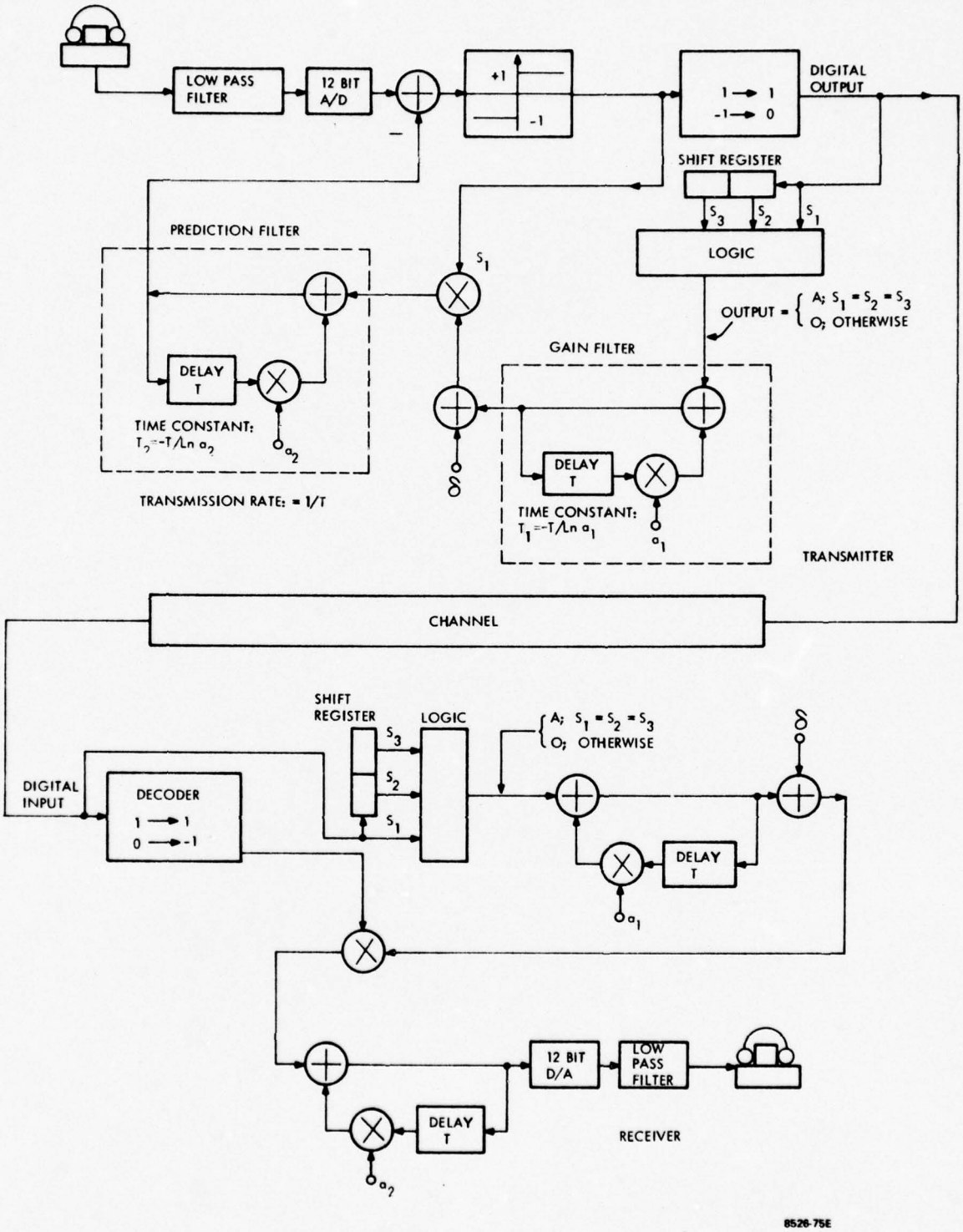


Figure 3-19. Transmitter and Receiver of CVSD-B

3.3.2Standard Three Bit Memory (CVSD-B)

Figure 3-19 shows the block diagram of the CVSD-B transmitter and receiver. By employing a three bit shift register, memories of the present and past two error samples are retained and utilized to estimate the incoming speech. Whenever the three bits of the shift register are identical (111 or 000), a waveform with steep slope is usually present. Then a spike of amplitude A is introduced into a single pole smoothing filter with coefficient a_1. Whenever the three bits are different, the logic introduced a zero into the same filter shown as:

$$f(j) = a_1 f(j-1) + \begin{cases} A & \text{, if shift register = 000 or 111} \\ 0 & \text{, otherwise} \end{cases} \qquad (3\text{-}1)$$

Then the filter output for a constant input amplitude A or 0 is given by

$$f(n) = \begin{cases} \frac{A}{1-a_1}\,(1-a_1^{n+1}) & \text{; if shift register = 000 or 111} \\ a_1^{n} & \text{; otherwise} \end{cases} \qquad (3\text{-}2)$$

When f(j) is added to a small bias δ, the result is a gain factor for multiplying the ± 1 data coming from the error quantizer. This new waveform then generates the speech waveform by exciting a second single pole integration filter with coefficient a_2.

In order to determine the amplitude of A that yields the best system performance, a parameter called compression ratio (CR) which is defined as

$$CR = \frac{f_{max} + \delta}{\delta} \quad \text{where } f_{max} = \frac{A}{1-a_1} \quad \ldots\ldots \qquad (3\text{-}3)$$

was investigated. This compression ratio is related to A by

$$A = \left(1-a_1\right) * \delta * (CR-1) \qquad (3\text{-}4)$$

With fixed δ and a_1, the amplitude of the spike A is directly proportional to the compression ratio. With a higher value of compression ratio, the system tracks the incoming waveform with a smaller slope overload.

As shown in Eq. (3-4), high pulse height A can also be obtained by raising the bias value δ. Hence a high δ tends to yield a better estimate to a fast changing input waveform. But in the presence of a slow varying waveform, such as silence intervals between speech, the main excitation component to the integration filter is δ which yields an oscillatory reconstructed waveform with a value $\pm \delta$. Hence large δ tends to increase the granularity of the processed speech.

Besides the above parameters, the time constants T_1 and T_2 of the gain and prediction filters also contribute to the reconstruction of speech waveform. A large value for the time constant T_1, which is related to the coefficient a_1 by

$$T_1 = -T/\ln a_1 \qquad (3\text{-}5)$$

Where $\frac{1}{T}$ is the transmission rate tends to have a smoothing effect on producing the quantizer gain which decreases the sensitivity to channel errors. A small value for time constant T_2 which is given by

$$T_2 = -T/\ln(a_2) \qquad (3\text{-}6)$$

produces a more energetic tracking of the input waveform.

3.3.3 Forney and Qureshi's 3 Bit Memory (CVSD-C)

Another strategy for updating the quantizer gain using exponentials is shown in Figure 3-20. This algorithm uses a 3-bit shift register with the last bit corresponding to the present error sample. The content of this shift register are used to generate pulses of different length which excite a smoothing filter. Table 3-1 provides the amplitudes of these pulses as a function of the shift register contents.

Content of Shift Register	Value
000	3/16
001	0
010	-5/128
011	5/64
100	5/64
101	-5/128
110	0
111	3/16

Table 3-1: Pulse Heights for FORNEY CVSD Quantizer

The output of this filter after addition of a small bias δ, is used as the exponent on the expansion 2^x. For implementation purposes, an approximation to the exponential expression is given by

$$2^{(IQK + QKF)} = 2^{(IQK)} \ (1+QKF) \qquad (3\text{-}7)$$

where IQK = integer part of the filter output and bias

QKF = fractional part of the filter output and bias

The result of this exponentiation is then used to excite an integration filter. Because exponentials are used for both increasing and decreasing the quantizer, a faster gain can be

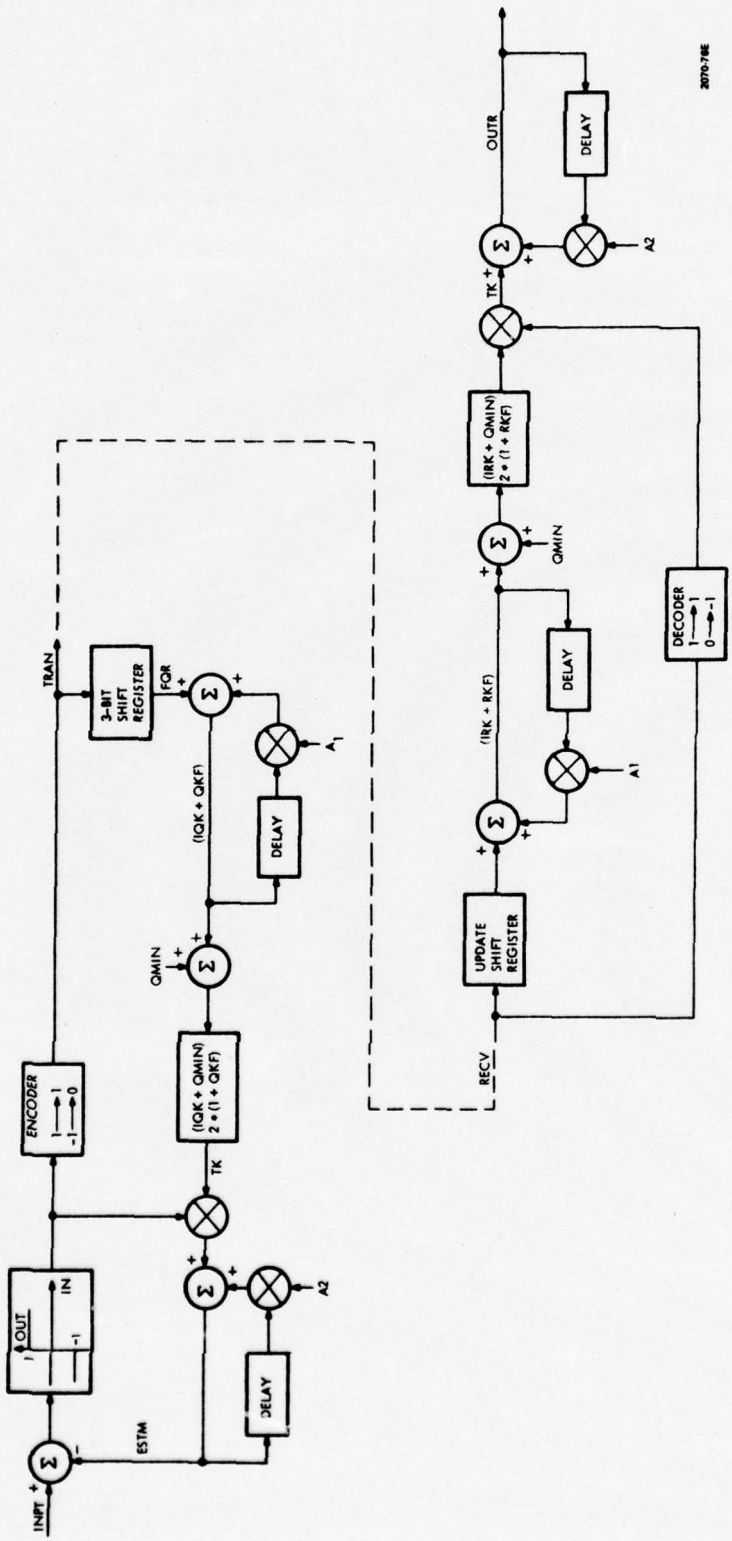


Figure 3-20. Block Diagram of CVSD-C

achieved within a relatively short time, resulting in a better dynamic range of the system and better tracking of rapidly increasing and decreasing waveforms.

3.3.4 Jayant's One Bit Memory (CVSD-D)

Figure 3-21 illustrates the Jayant slope adaptation logic.

In contrast to the two previous algorithms, the Jayant method uses only one bit of memory. Every time an error sample is generated, it is compared with the previous quantized value. If they are different, the present quantizer level is multiplied by a factor $Q < 1$. If they are the same, it implies a continuous rising or falling waveform. Then the present level is multiplied by a factor $P > 1$. They are used to excite the integration filter to obtain an estimate of the incoming speech waveform.

3.4 Discussion of Real-Time Code

The real time software of CVSD is straightforward when compared to APCQ program codes. The sequence of operations involve the initialization of speech and line side interrupts, speech processing using one of the quantizer gain update strategies as discussed in Section 3.2, transmission of error code and synthesis of speech afterwards.

As before, the two interrupts, namely the speech and lineside interrupts are used. The former obtains its signal from a 12-bit A/D at a constant rate and then output a 12 bit processed sample for a 12-bit D/A. The latter governs the transmission of data. At every interrupt, it outputs a transmitter clock pulse and decodes one bit of received information. At every 8 interrupts, it inputs a new eight bit receiver word. The program transfers control to a wait loop after it finishes performing either transmitter or receiver functions. Since there is no framing necessary in the CVSD algorithm, synchronization is not needed.

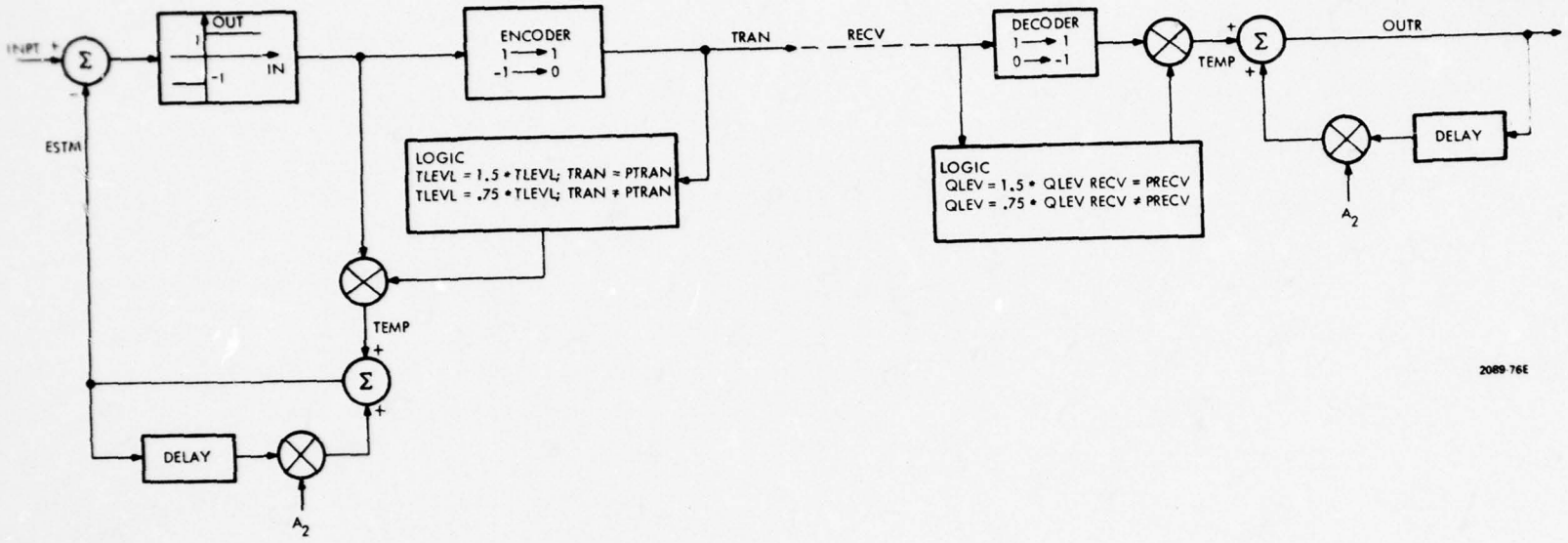


Figure 3-21. Block Diagram of CVSD-D

3.4.1 Wait Loop

Figure 3-22 shows a flow chart of the wait loop. Its function, as mentioned earlier, is to use up all the time left over after processing operations. It also sequences the transmitter and receiver operations by examining the transmitter and receiver ready flops and examines the front panel to see if initialization is desired.

3.4.2 The Line Side and Speech Side Interrupts

Figures 3-23 and 3-24 describe the software operations when interrupts occur. The function of these interrupts are discussed in Sections 3-14 and 3-15.

3.4.3 Speech Analysis and Synthesis

Figures 3-25 and 3-26 describe the software operation of the transmitter and receiver of CVSD-B. It allows a selection of transmission rates (9.6 kBPS or 16.0 kBPS), time constants in gain filter (6.3 msec, 10 msec, 20 msec), time constants in prediction filters (1 msec, 2 msec, 10 msec, 20 msec) and compression ratios (10, 20, 60, 100, 150, 200). Figure 3-27 and 3-28 show the software operation of Forney and Qreshi's CVSD-C. Figures 3-29 and 3-30 describe the operation of CVSD-D transmitter and receiver where one bit of memory is used.

3.5 Operating Procedures of CVSD Program

3.5.1 Connecting the System

Connect the EDM's to each other or through a modem as detailed in Section 3.2.1.

3.5.2 Reading in the EDM Software

Read in the CVSD program as shown in Section 3.2.2.

3.5.3 Initialization of the EDM

After the program has been executing, reinitialization can be achieved by simultaneously holding switch register toggle switch 15

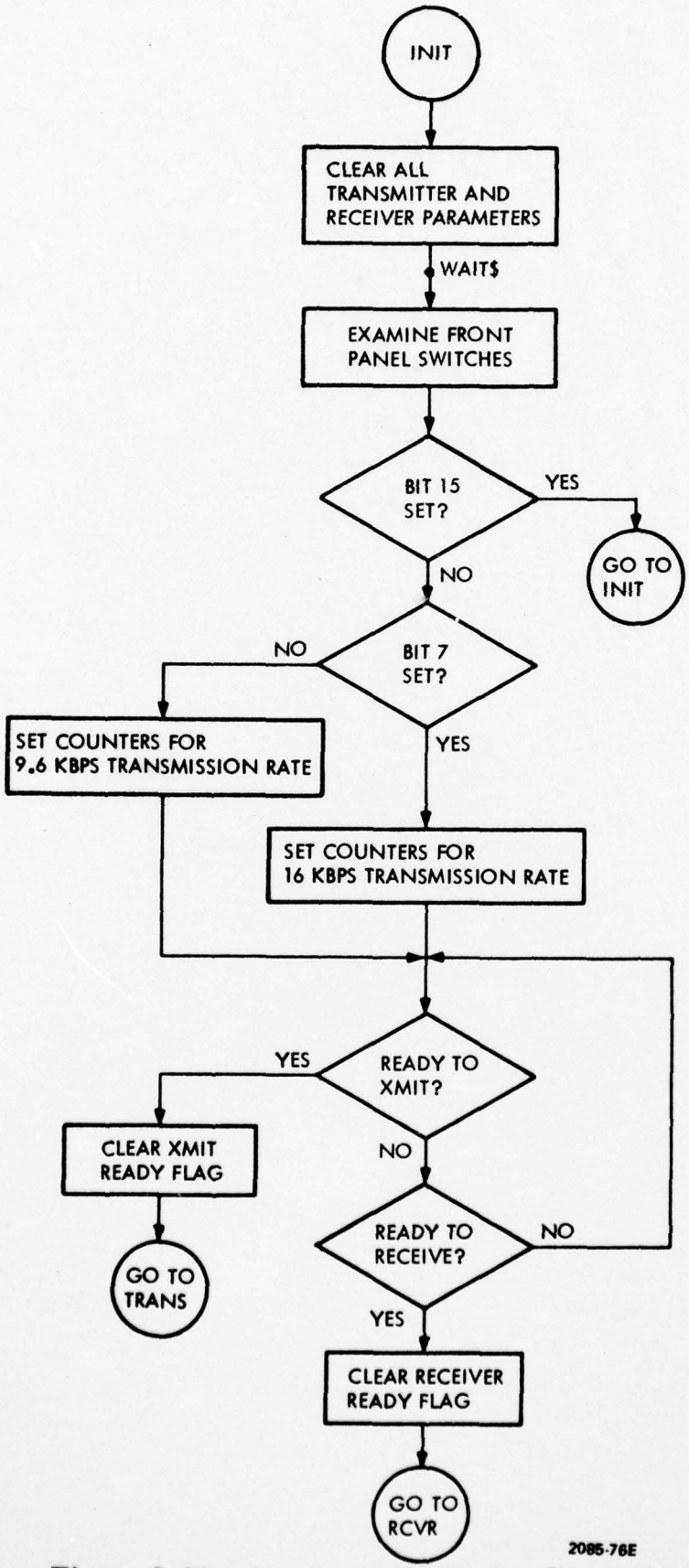


Figure 3-22. Initialization and Wait Loop

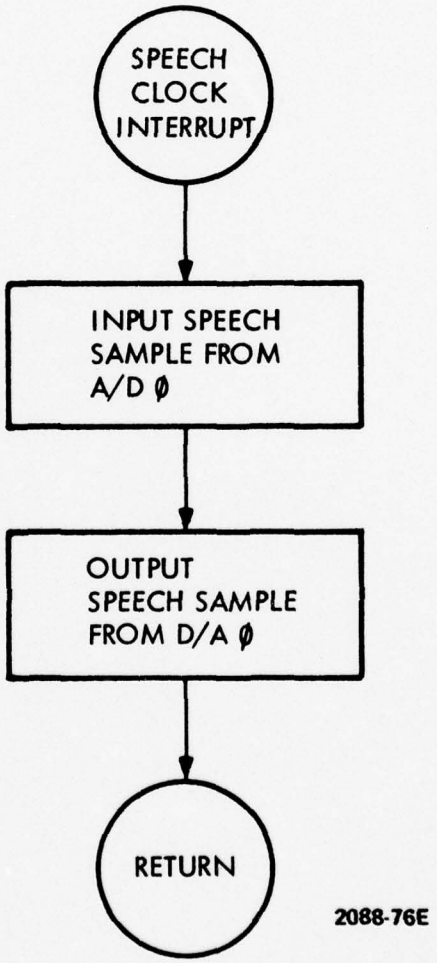


Figure 3-23. Interrupt Loop Speech Side

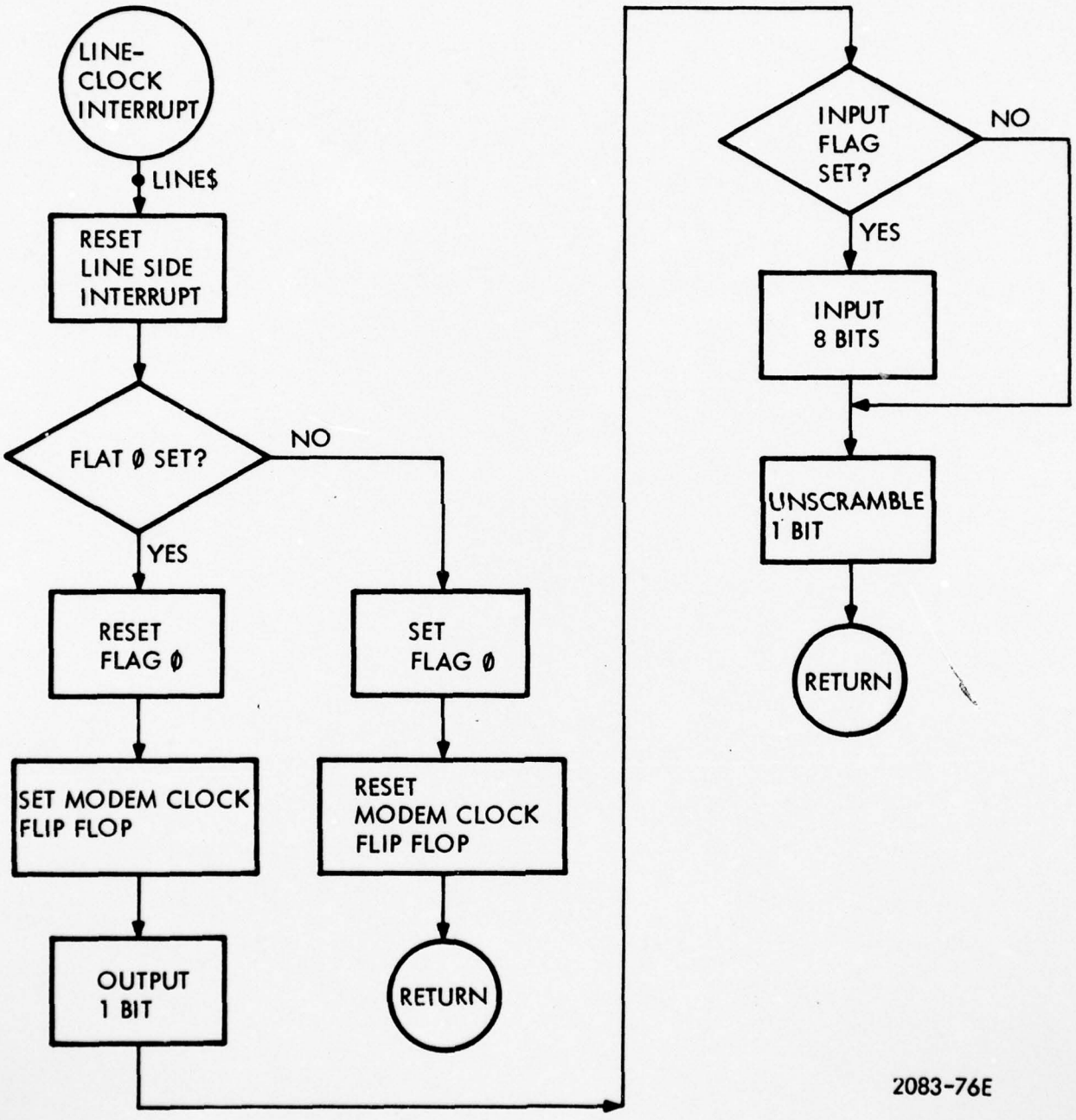


Figure 3-24. Interrupt Loop Line Side

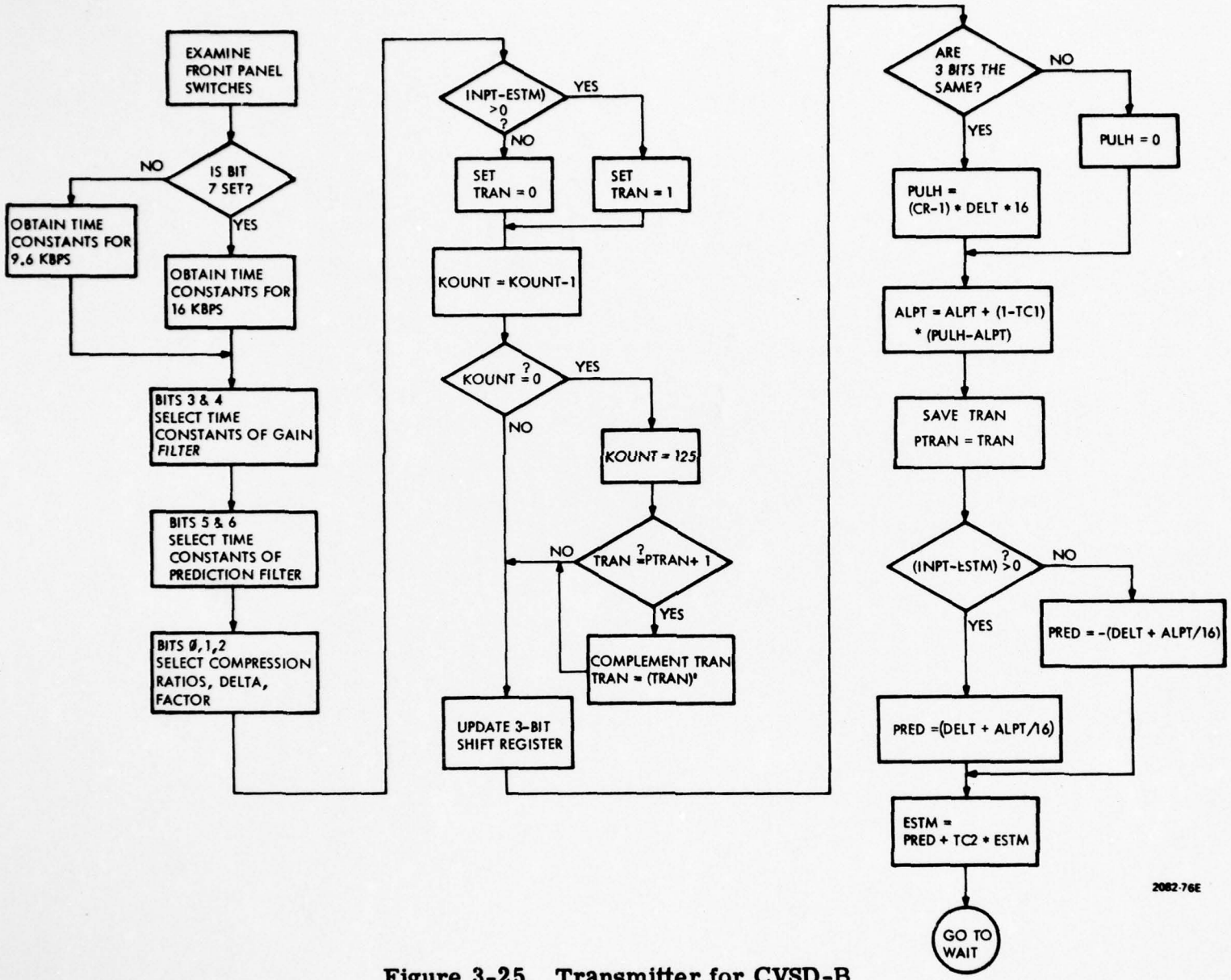


Figure 3-25. Transmitter for CVSD-B

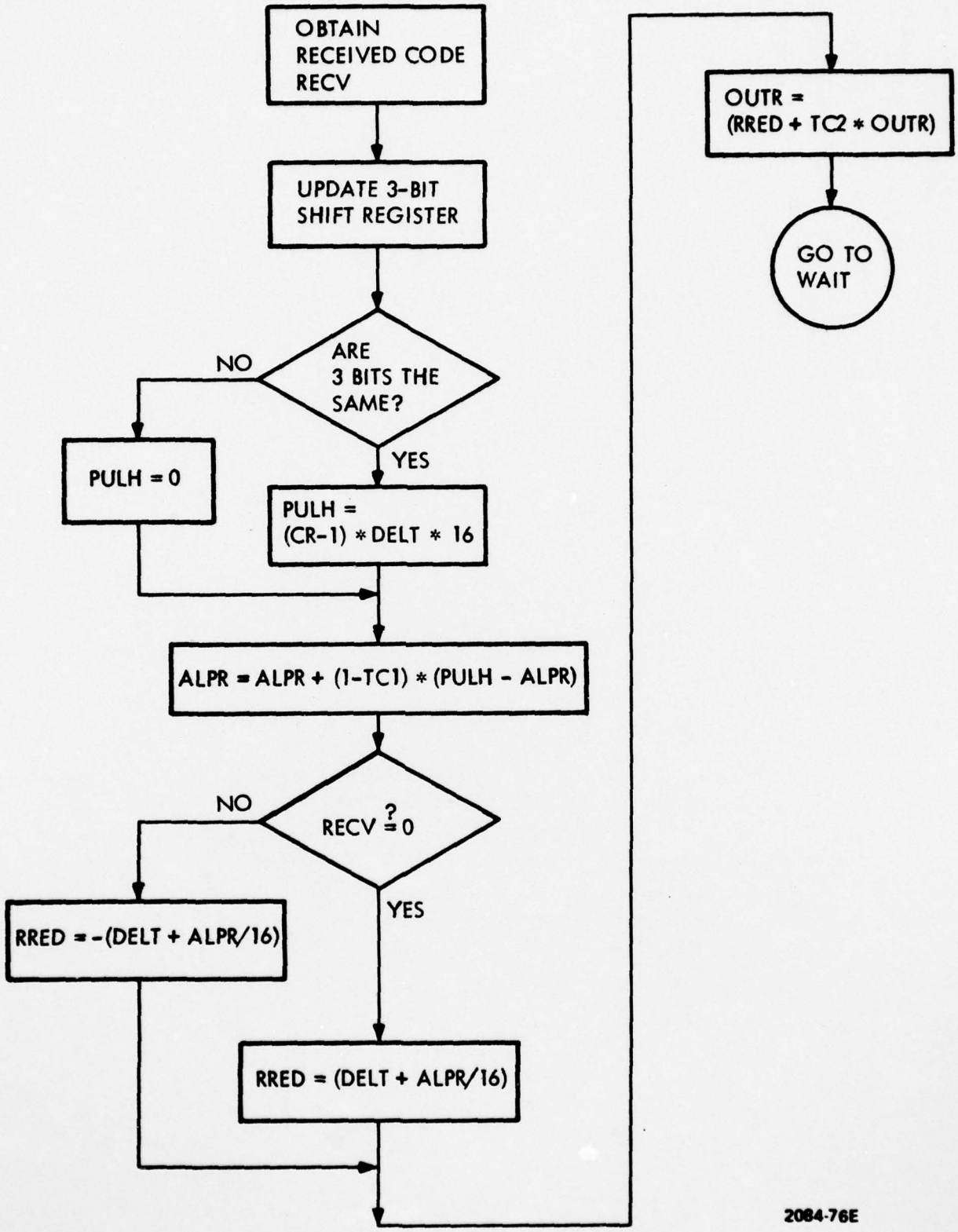


Figure 3-26. Receiver for CVSD-B

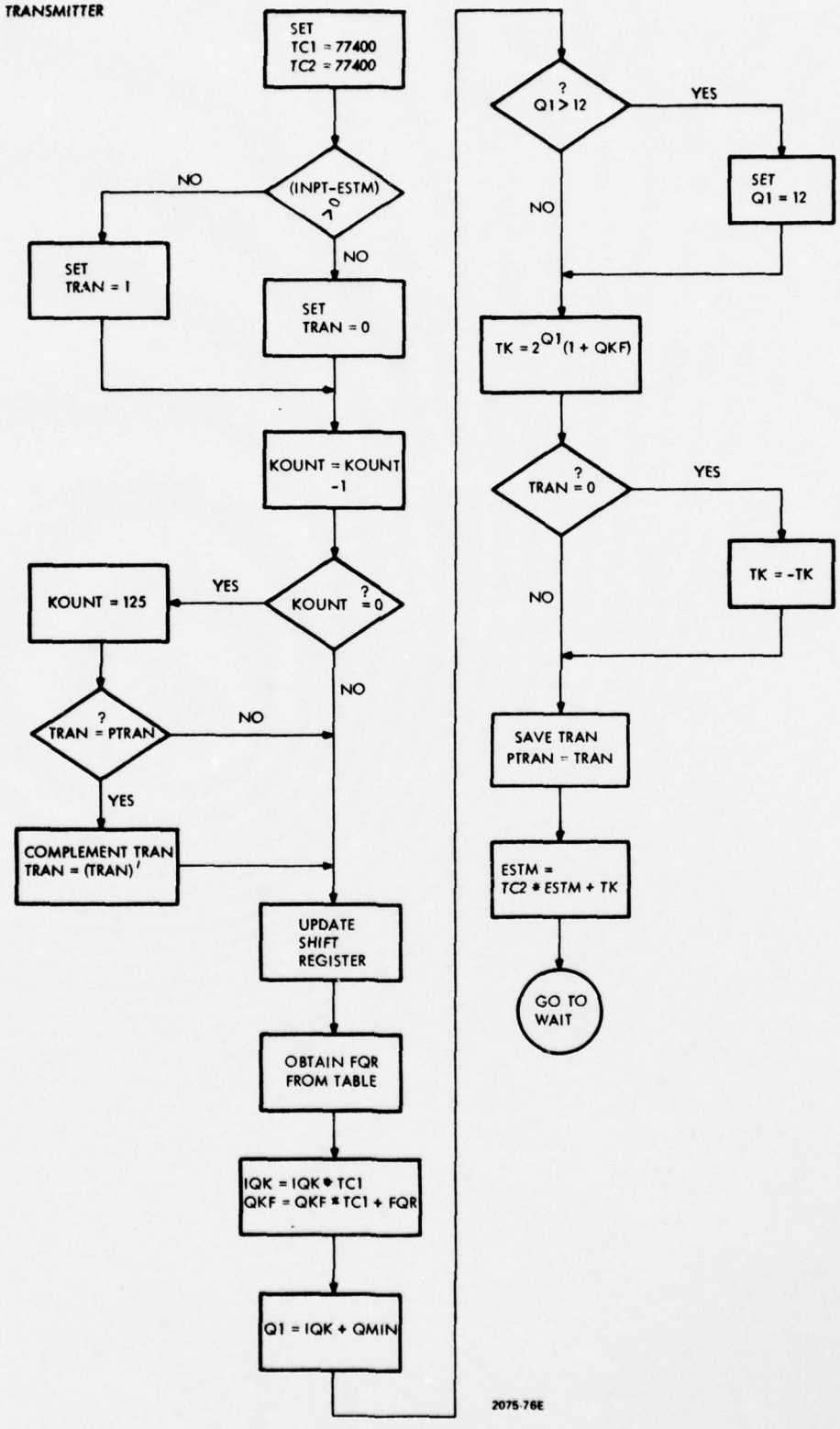


Figure 3-27. Transmitter for CVSD-C

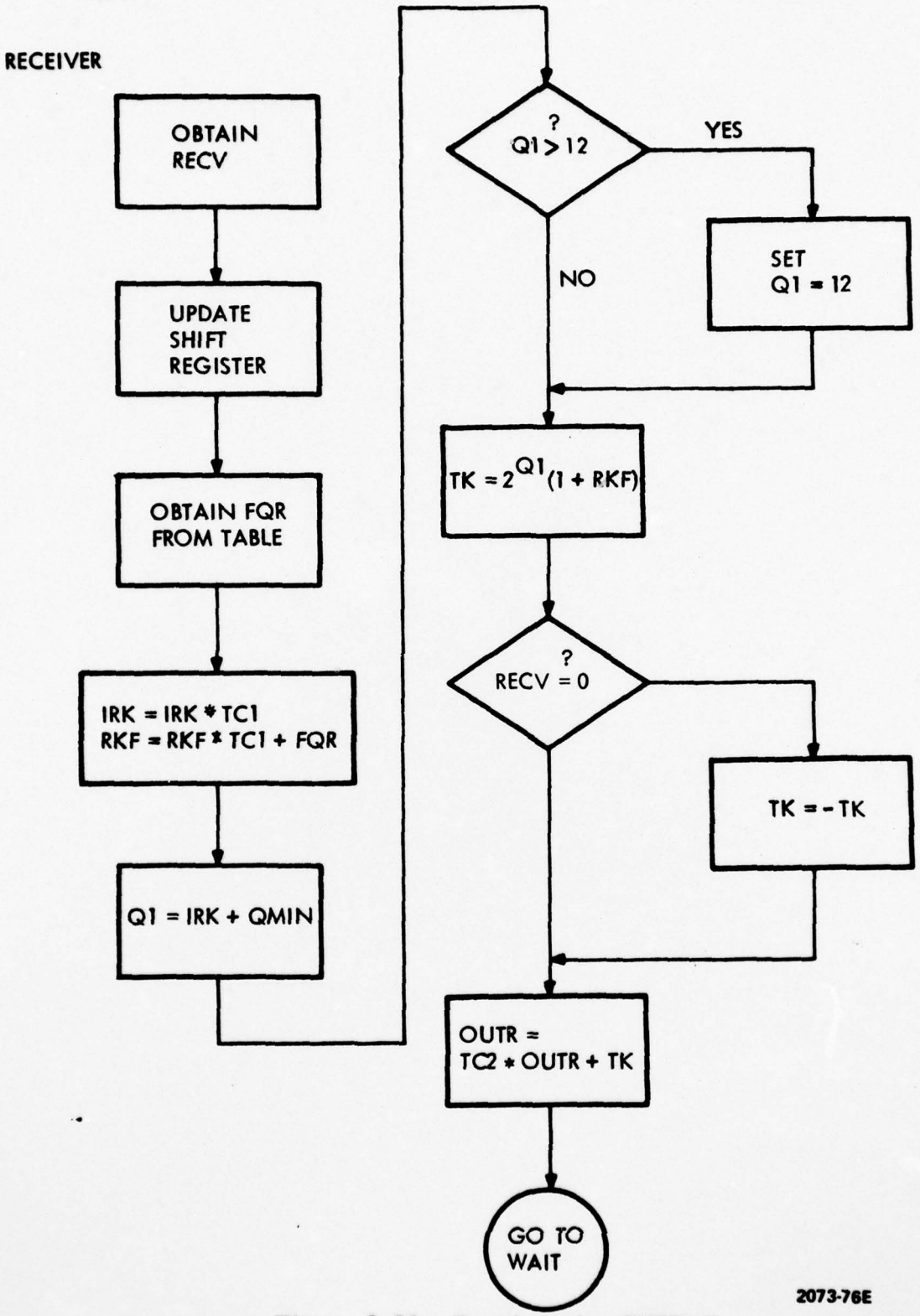


Figure 3-28. Receiver for CVSD-C

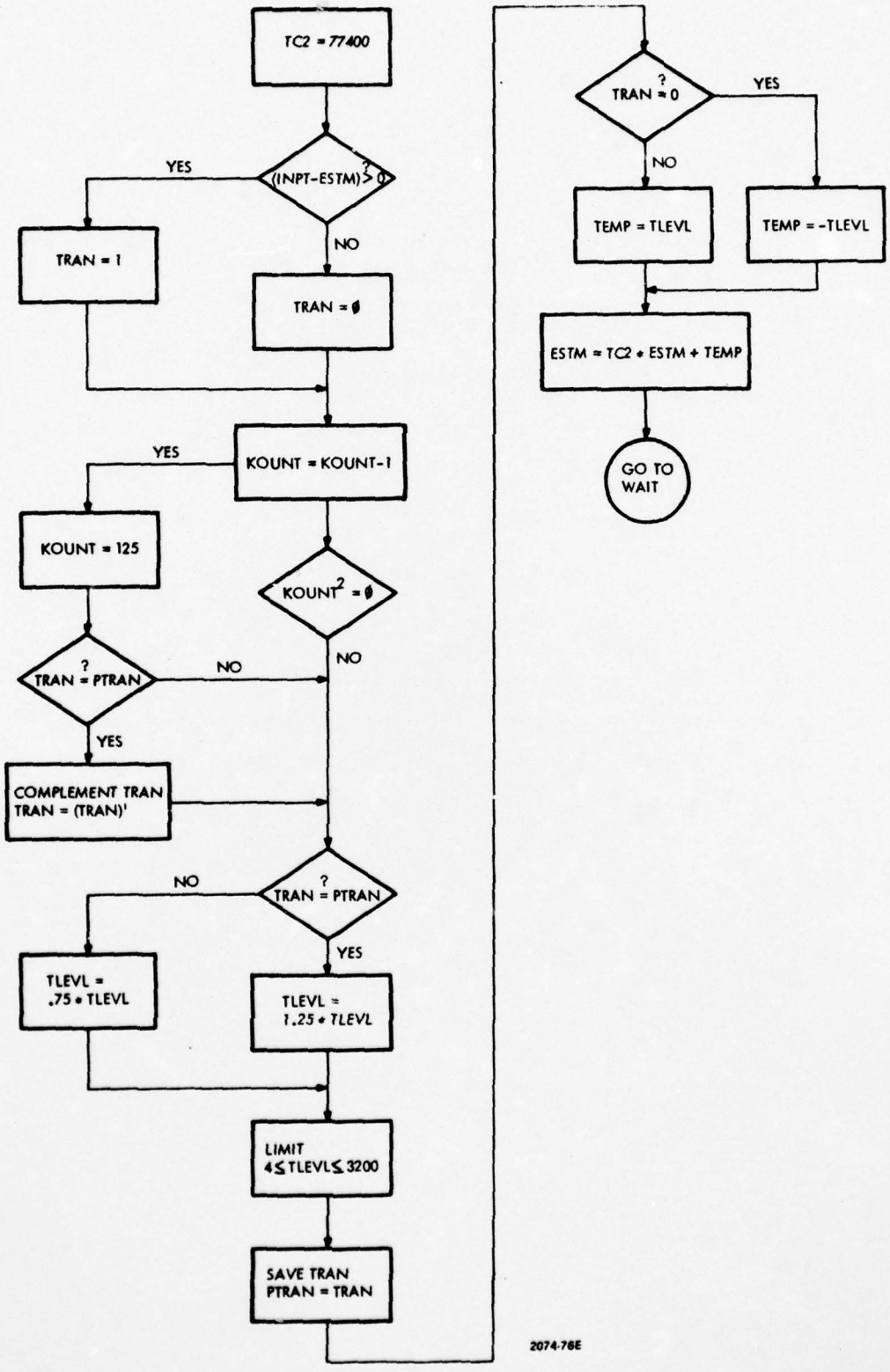


Figure 3-29. Transmitter for CVSD-D

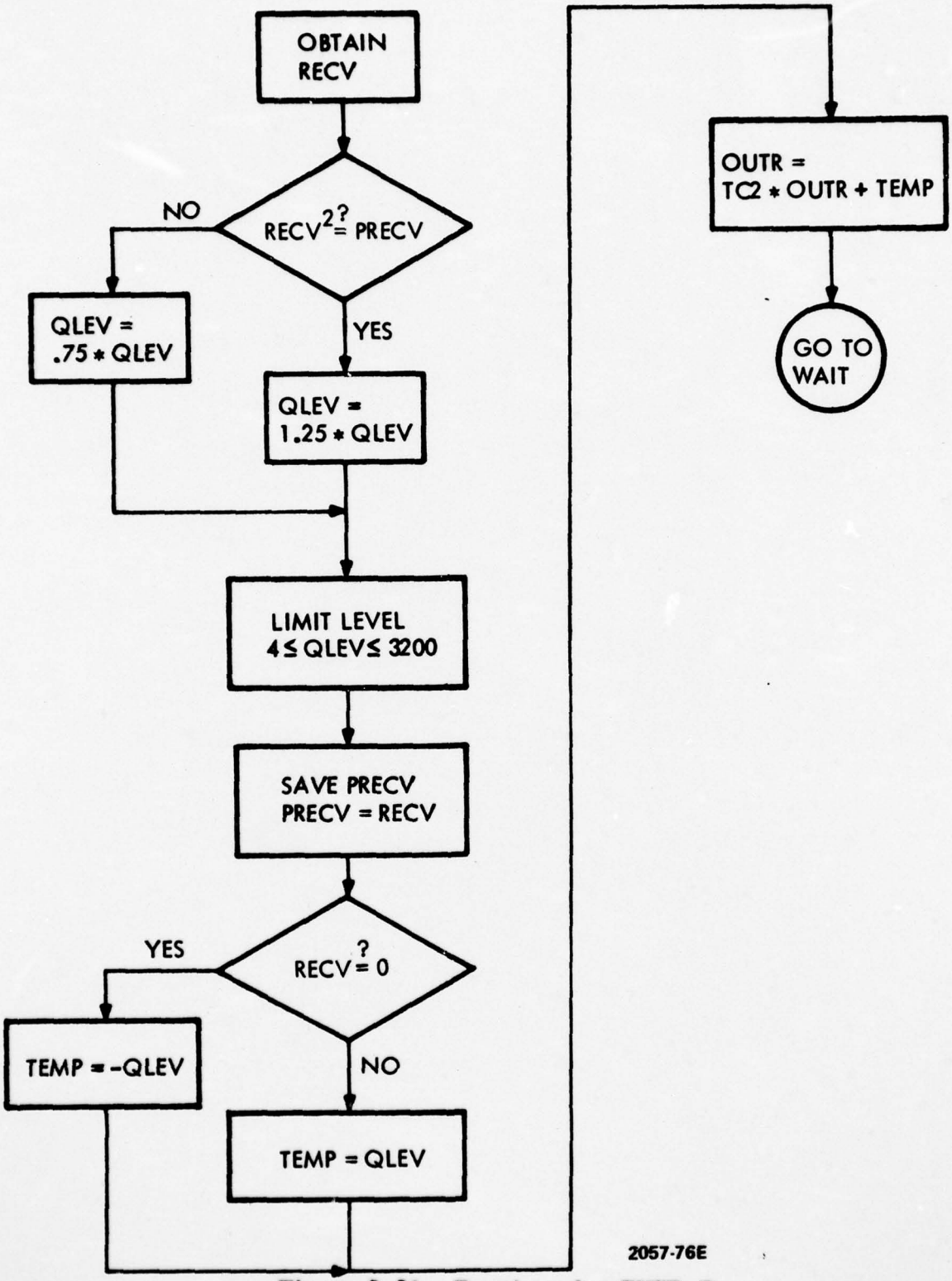


Figure 3-30. Receiver for CVSD-D

in the down position for each EDM and then returning it to the up position.

3.5.4 Program Options

The CVSD program enables selections of two transmission rates, three slope adaptation logics, six compression ratios, three time constants for gain filter and four time constants for the prediction filter. The EDM switch settings and their functions are summarized as follows:

i) Initialization

BIT 15 down = initialization
up = program execution

ii) Transmission Rates

BIT 7 down = 16.0 kBPS
up = 9.6 kBPS

iii) Slope adaptation logics

a) BIT 8 and BIT 9 up = standard 3-bit (CVSD-B)

BIT 2	BIT 1	BIT 0	Compression Ratios
0	0	0	10
0	0	1	20
0	1	0	60
0	1	1	100
1	0	0	150
1	0	1	200

BIT 4	BIT 3	Gain Filter Time Constant
0	0	6.3 msec
0	1	10.0 msec
1	0	20.0 msec

BIT 6	BIT 5	Prediction Filter Time Constant
0	0	1.0 msec
0	1	2.0 msec
1	0	10.0 msec
1	1	20.0 msec

b) BIT 8 down and BIT 9 up = Forney and Qreshi's 3-BIT (CVSD-C)

c) BIT 8 up and BIT 9 down = Jayant's 1-bit (CVSD-D)

3.6 I/O Speech Filters

Under this contract, GTE Sylvania supplied 4 identical I/O Speech Filters, two for each DCA PSP. The I/O speech filters are passive, low-pass elliptic filters that exhibit an essentially flat passband characteristic (0-3200 Hz at less than 0.5 dB ripple) followed by a very sharp transition region occuring just below their 3 dB attenuation frequency. These filters, supplied by ESC Electronics Corporation, Pallisades, N.J., are very stable and possess a very low noise figure as a result of their passive design. The mechanical and electrical specifications insure compatibility with the DCA PSP's and they can be plugged into card number 1 of the PSP without any modifications. Figure 3-31 provides the frequency loss characteristics of the filter (ESC part number ESC43F61) whose input and output impedance is 5000 ohms. These filters have a pin arrangement assure that will be inserted correctly into the PC board. Care should taken, however, to not damage the filter pins when inserting or removing the filter from the card.

AD-A031 516 GTE SYLVANIA INC NEEDHAM HEIGHTS MASS ELECTRONIC SYS--ETC F/G 17/2
ADAPTIVE MULTILEVEL 16KB/S SPEECH CODER.(U)
JUN 76 A GOLDBERG DCA100-76-C-0002

UNCLASSIFIED NL

2 OF 2
ADA031516

END

DATE
FILMED
12 - 76

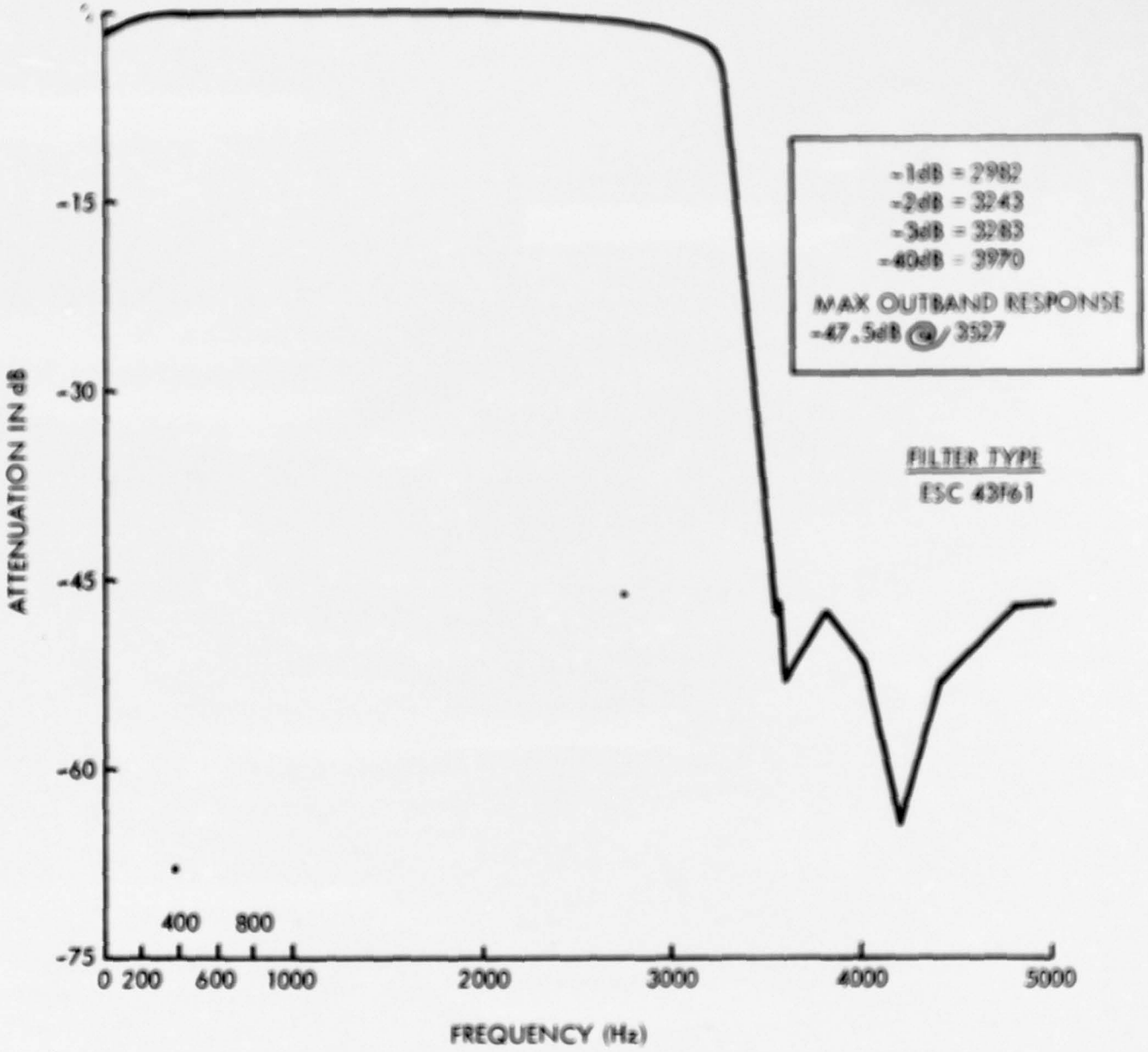


Figure 3-31. I/O Filter Characteristics

SECTION 4

LSI Implementation Study

4.1 Introduction

Because of the digital signal processing involved in the APCQ technique, its implementation should be all digital. Today large scale integration (LSI) fabrication techniques provide digital designs having lower cost, smaller size, greater reliability and less power consumption.

Under this contract, GTE Sylvania examined different LSI devices and organized them into a signal processor suitable for APCQ implementation. The processor design has a modular, bus-oriented architecture with highly program controlled parallelism.

The form of design will provide an implementation of the APCQ coder which uses about 60% of the processing time available. Our estimates indicate that the APCQ algorithm will require about 16 msec. to analyze speech and to synthesize speech during each 25 msec. data frame. Thus, the processor hardware idles about 9 msec. out of each frame. This idle time can be used for other telephone subscriber functions, such as encryption and telephone line control.

The cost of parts is also surprisingly low. Today's cost for parts in small quantities is $700 and their power requirements are just over 42 watts. Thus, the APCQ coder, while mathematically complex, can be built today using commercially available LSI processors for a relatively low cost.

The next section will present the architecture of the design followed by a discussion of the LSI software implementation of the APCQ algorithm. Estimates of timing, memory, cost and power

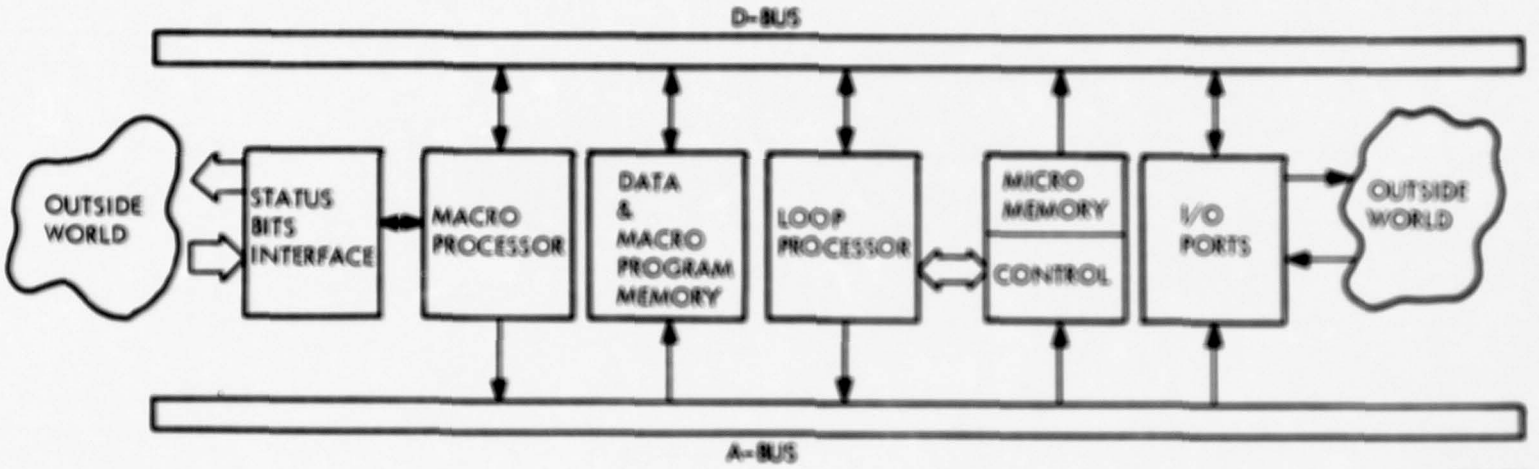


Figure 4-1. Architectural Concept

are then presented in Section 4.3

The processing through out of the design can certainly be increased by adding higher degrees of parallelism. In Section 4.4, we illustrate how adding an index processor or an external hardware multiplier or both can improve performance through increased hardware complexity and through a rearrangement of the busing structure. The estimates of these different designs are included as a comparison.

4.2 Basic LSI Signal Processor Architecture

The basic LSI signal processor Architecture for APCQ implementation consists of a macro processor, a loop processor, memory and peripheral devices. They are interconnected via two main buses as shown in Figure 4-1.

Each of the two processors is designed to perform the tasks for which it is best suited: the macro processor for setup and program control, the loop processor (LP) for array processing in 16-bit or greater precision and the handling of interrupts at a high rate.

The macro processor is a mini-computer on a chip. The Texas Instrument TMS 9900 is chosen as this 16-bit processor. It is designed to be capable of running at a 3 MHz micro-cycle rate.

The 16-bit main memory contains both the working storage (both constants and variables) for the macro processor and the LP and the program memory for the macro processor.

The loop processor is the fast arithmetic and logic unit. As shown in Figure 4-2, it consists of an A Register to interface with the main memory and the powerful ALU chips, which are

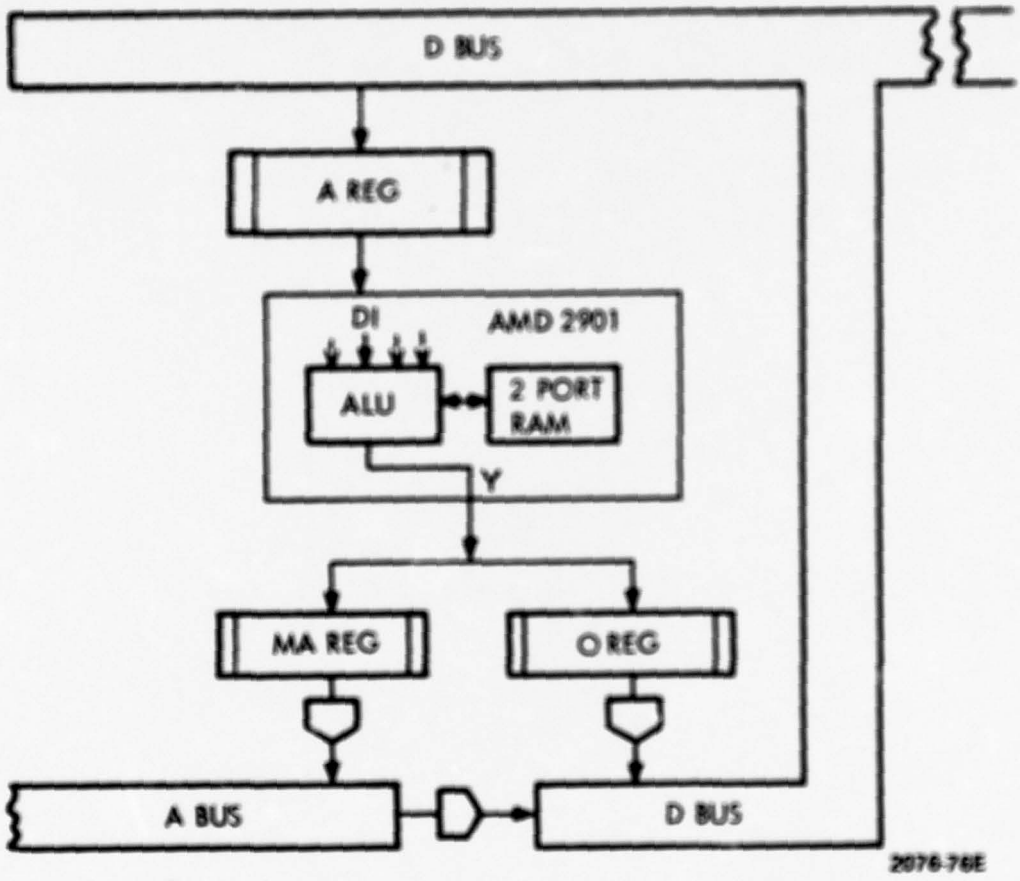


Figure 4-2. Loop Processor

the Advanced Micro Devices AM2901 4-bit slice mirco processors.

Output of the ALU chip can be either latched through the output Register (O register) to the data bus (D Bus) or through memory address register (MA register) to the address bus (A Bus). Additional A Bus to D Bus connection is provided in the design for added memory access flexibility.

The micro program memory and control hardware governs the activities of the LP. The control hardware includes micro program controlled storage of ALU status, micro jump control, subroutine nesting and priority interrupt handling.

Both interrupting and non-interrupting I/O ports are used. Each port can be individually addressed and commanded via the A Bus, either placing data onto the D Bus or accepting data from it. Interrupting ports issue service requests to the LP, which controls the buffering of input or output data on a priority interrupt basis. Non-interrupting ports can be accessed at any time by either the macro processor or the LP.

All bus-driving points use three-state output devices widely available in current technology at bipolar speeds. All devices in our design are off-the-shelf LSI, MSI and SSI Schottky bipolar chips.

The main ALU, the AM 2901, is believed to be the most powerful micro processor chip available at the present time. It meets all military specifications and can function at a clock rate of 6.66 MHz, or 150 ns cycle time.

One key feature of the processor is its 16 word 2-port RAM. The RAM serves as an on-chip register file. 2-port makes it

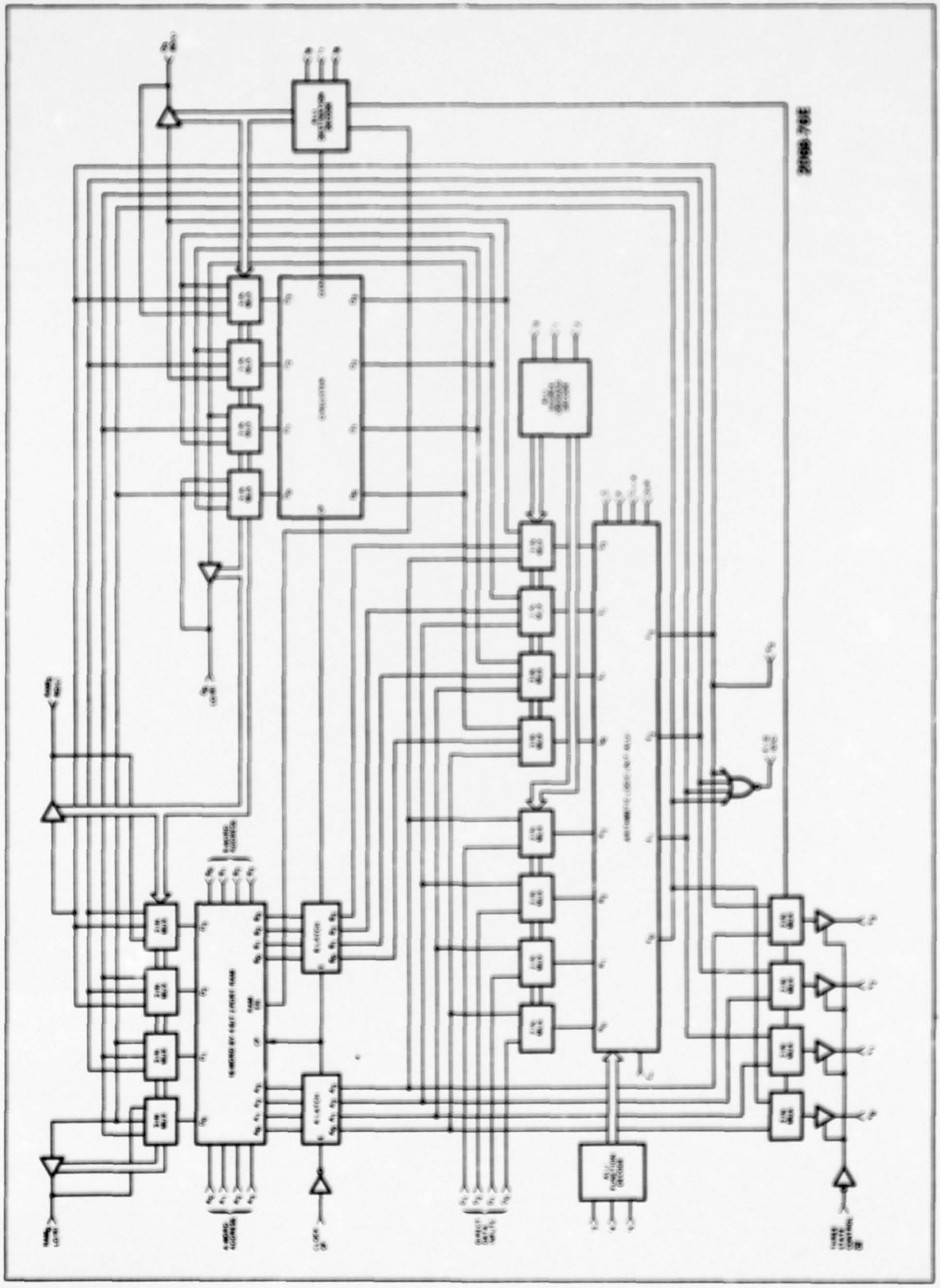


Figure 4-3. Detailed AMD2901 Microprocessor Block Diagram

easy to access two registers at the same time. In addition, there is the Q register, which is intended primarily for multiplication and division routines. The detailed AM 2901 block diagram is shown in Figure 4-1.

The signal flow of the processor shown in Figure 4-2 can be explained as following. Input data are fed from data memory to the D Bus (Data Bus) and latched by the A register. They can either be saved in one of the locations in the 2-port RAM or directly feed the ALU, which does the arithmetic or logical operations. Y output from the ALU can be fed back to the processor for further ALU manipulation or can be latched to the Q register or the MA register. This data can, therefore, be brought back to data memory via the D Bus or used as a memory address pointer to access new data from memory.

The branching hardware, which is not shown in Figure 4-2, looks at the result of the previous cycle. All branches are executed at the end of the cycle following the instruction containing the branch. (This is the result of a pipelined instruction fetch). All the operations are controlled by the micro program instructions and synchronized to 13.33 MHz single phase clock.

4.3 Estimates of LSI Implementation of APCQ Algorithm

This section presents the transmitter operations of the APCQ algorithm. All sections of the transmitter are analyzed and some are shown as examples. Overall timing and memory estimates of the entire APCQ implementation are supplied. Finally, the cost and power of the LSI processor based on counted chips are documented.

4.3.1 APCQ Implementation

LSI design implementation of APCQ Algorithm is based on

the PSP implementation of the same algorithm. The transmitter operations are listed in Figure 4-4.

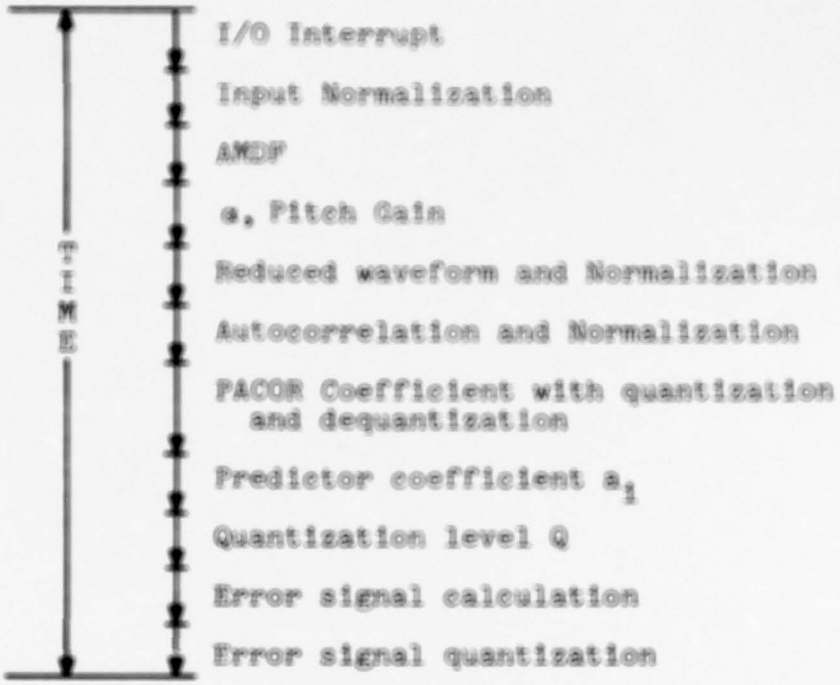


Figure 4-4 APCQ Transmitter Operations

Each task in the transmitter is implemented as separate software for execution in the loop processor (LP). Many of the software tasks involve repeated calculations on data arrays. These calculations are performed in software loops. The macro processor will handle the loop and interrupt control. Therefore, the overall timing and memory can be estimated as those required in the LP, the macro processor and some overhead.

The use of the macro processor is presented conceptionally in Figure 4-5 along with the LP operational framework and memory map. Within this organization, the macro processor acts as the master computer, and the LP as a programmable peripheral processor for doing the complex computations required by the processing algorithm.

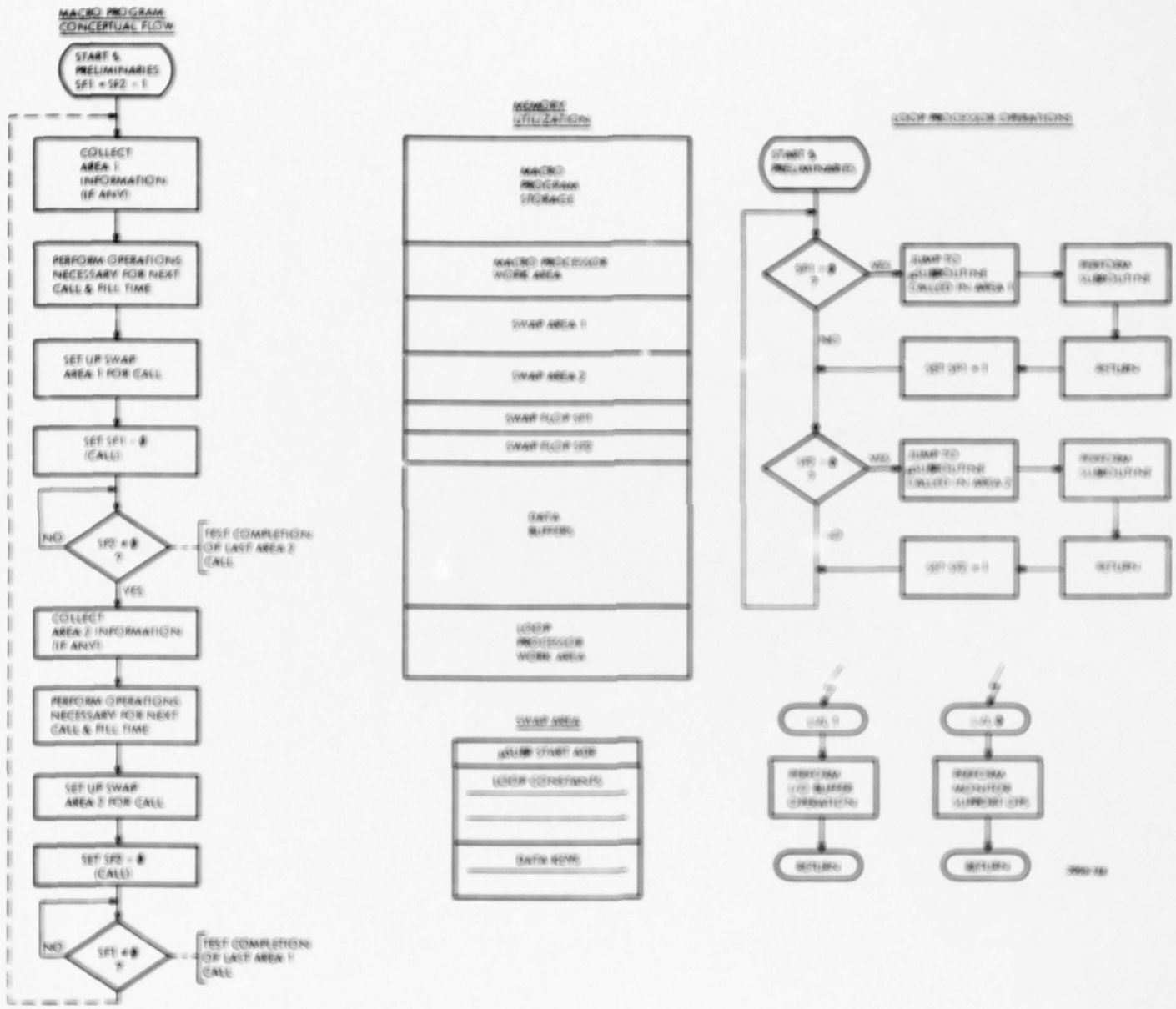


Figure 4-5. Operational Flow Macroprocessor and Loop Processor

This technique of resource use hinges on the use of two "swap areas" in memory, accessed by one processor at a time, with software control of access via two "swap flops". Initially, both "swap flops" are set to 1 and the macro processor is granted access. The LP simply spins in a loop waiting for one of the SF's to become Ø.

The macro processor then sets up for a loop processor "CALL" by placing subroutine parameters into a swap area and clearing the associated SF. The LP (when finished with processing in the other area, if any) will then access the swap area and perform the called routine.

The macro processor (after checking for access to the remaining swap area) then sets up for the next LP CALL and clears that "swap flop".

Note that the macro processor always does all the work it can, until both swap areas are given over to the LP and, once given access to a swap area, the macro processor accepts data from it (when appropriate) before setting up the next LP CALL.

The LP, in its turn, simply obeys the CALL when given access to a "swap area", completing the subroutine (such as an AMDF or Autocorrelation) and then returning access to the macro processor.

Two levels of interrupt processing are indicated at the LP, one for INPUT/OUTPUT service and one to support data transfer to the monitor interface for test and debugging operations. These operations are assigned to the LP due to its speed in the first case and due to its ready access to fast registers in the second case.

This framework provides for full resource use within the confines of any given organization of a processing algorithm with both the macro processor and loop processor working simultaneously whenever possible. With this protocol as background, we proceed below to concentrate on the LP subroutines for the APCQ algorithm.

4.3.2 Examples

The functional Activity Diagram for the Loop Processor is used to analyze the APCQ implementation. All transmitter operations of the APCQ coder were analyzed and examples of the more time consuming tasks are provided on the following pages.

a. Autocorrelation Function

Figure 4-6 is the functional activity diagram of the inner loop of the ith autocorrelation. Software multiplication is initialized by loading the multiplier into the Q register and the multiplicand into R_3. The product appears in R_1 and Q. The 32-bit result is accumulated in double precision in location R_4 and R_5. CO represents the carry out of the low-order accumulation. R_{10} is used as a loop control by presetting it to 128.

The outer computation loop, which is not shown for the sake of clarity, performs the functions of storing R_4, R_5, increasing R_2 up to maximum, setting $R_\emptyset$, R_{10}, R_2, etc.

The outer loop can be performed in 15 machine cycles by the LP. The inner loop takes 22 cycles. Therefore, the total processing cycle for this function is:

$$22 \times 128 \times 5 + 15 \times 5 = 14155 \text{ cycles}$$

Autocorrelation will also be normalized such that $C_0=1$. Thus, four normalizations will require 4 multiplications and 1 division.

The total time to calculate the autocorrelation functions plus normalization is:

$$14155 + 5 \times 22 + 132 = 14397 \text{ cycles}$$

b. AMDF

Figure 4-7 diagrams the inner loop of the AMDF. R_5 is used to increase the memory address by three. R_1 and R_2 are initialized at the memory address index. The results are accumulated in register pair R_3 and

PROGRAM SEQUENCE	Arithmetic Logic Unit				BRANCHING
	A REG	MA REG	O REG	FUNCTION	
1		R0		R0 → MA, R0 -1 → R0	
2	X_i			R0 + R_2 → MA	
3	X_{i+j-1}			D → R_3	
4				D $\xrightarrow{SRA}$ Q, R_3 $\xrightarrow{SRA}$ R_3	16 cycles multiplication
. .				. .	
. .				. .	
19				Result in R_1, Q	
20				R_{10} - 1 → R_{10}	
21				R_4 + R_1 → R_4	Branch to 1
22				R_5 + CO → R_5	Branch exit if R_{10} = 0

$$\text{Autocorrelation } C_j = \sum_{i=1}^{128} X_i * X_{i+j-1} \quad j=0,1,2,3,4$$

Figure 4-6. Autocorrelation Function Inner Loop

PROGRAM SEQUENCE	Arithmetic Logic Unit				BRANCHING
	D	MA REG	O REG	FUNCTION	
1				$R_7 - 1 \rightarrow R_7$	
2		i		$R_1 + R_5 \rightarrow R_1$, MA	
3	S_i	i-t		$R_2 + R_5 \rightarrow R_2$, MA	
4	S_{i-t}			$D \rightarrow R0$	
5				$R0 - D \rightarrow R0$ SAVE SIGN IN L	
6		i		$R_1 + R_5 \rightarrow R_1$, MA	
7	S_i	i-t		$R_2 + R_5 \rightarrow R_2$, MA	
8	S_{i-t}			$D \rightarrow R_4$	Branch to 16 if L=1
9				$R_4 - D \rightarrow R_4$ SAVE SIGN IN Z	
10				$R_3 + R_0 \rightarrow R_3$	Branch to 13 if Z=1
11				$R_6 + CO \rightarrow R_6$	
12				$R_3 + R_4 \rightarrow R_3$	Branch to 1
13				$R_6 + CO \rightarrow R_6$	Branch to exit if R_7=0
14				$R_3 - R_4 \rightarrow R_3$	Branch to 1
15				$R_6 + CO \rightarrow R_6$	Branch to exit if R_7=0
16				$R_3 - R0 \rightarrow R0$	Branch to 13 if Z=1
17				$R_6 + CO \rightarrow R_6$	
18				$R_3 + R_4 \rightarrow R_3$	Branch to 1
19				$R_6 + CO \rightarrow R_6$	Branch exit if R_7=0

$$\text{AMDF:} \quad A_t = \sum_{i=1,3\ldots}^{72} \left| S_i - S_{i-t} \right| \qquad t = +15, \cdots, 45$$

Figure 4-7. AMDF Inner Loop

R_6 for double precision addition. CO represents the carry out of the lower sum. The loop computes and accumulates two absolute differences, therefore, it must be employed 12 times for each of 60 AMDF points. R_7 is used as a loop control, it is preset at 12.

The inner loop requires 13 cycles per iteration. The outer loop for the AMDF must reset the memory address pointer, store results etc.; it requires nine machine cycles. Therefore, the LSI processor can implement the AMDF in:

$$13 \times 12 \times 60 + 9 \times 60 = 9900 \text{ cycles}$$

c. The Reduced Waveform

The speech signal is filtered by a one-tap filter using the pitch gain, α, as the tap coefficient. Figure 4-8 is the functional activity diagram of the inner loop of the reduced waveform. R_2 is the memory address pointer to pick up the pitch gain, α. R_1, R_0, R_4 are also used as pointers to pick up data from memory or store data back into memory. The inner loop takes 24 cycles to perform the function. 15 more cycles are necessary to initialize the memory pointing register and some other overhead operations. Therefore, total required cycles are:

$$15 + 24 \times 128 = 3087$$

d. The Error Signal

Figure 4-9 is the functional activity diagram of the inner loop of error signals calculation. R_7, R_{10} and R_{11} are used to accumulate the results. The registers R_0, R_1, R_3, R_4 are used as memory address

PROGRAM SEQUENCE	Arithmetic Logic Unit				BRANCHING
	D	MA REG	O REG	FUNCTION	
1		n-M		R_1 - R0 → MA	
2	S_{n-m}	R_2		$R_2 \rightarrow R_2$, MA	
3	α			$S_{n-M} \rightarrow R_3$	
4				$\alpha \xrightarrow{SRA} Q$ $R_3 \xrightarrow{SRA} R_3$	16 cycles multiplication $\alpha * S_{n-M}$
⋮				⋮	
19				Result in R0, Q	
20		R_1		$R_1 \rightarrow$ MA $R_1 - 1 \rightarrow R_1$	
21	S_n			$R_5 - 1 \rightarrow R_5$	
22			S_n - R0	S_n - R0 → O	
23		R_4		$R_4 - 1 \rightarrow R_4$, $R_4 \rightarrow$ MA	
24				$O \rightarrow M_{(MA)}$	Branch 2
25		n - M - 1		R_1 - R0 - 1 → MA	Branch exit IF $R_5 = 0$

$$V_n = S_n - \alpha S_{n-M} \qquad n = 1, 2 \cdots 128$$

Figure 4-8. Reduced Form Inner Loop

PROGRAM SEQUENCE	Arithmetic Logic Unit				BRANCHING
	D	MA REG	Q REG	FUNCTION	
1		R0		$R_0 \rightarrow MA,\ R_0$	
2	a_i	$R_1 - R_0$		$R_1 - R_0 \rightarrow M_A$	
3	V_{n-i}			$a_i \rightarrow Q$	
4				$V_{n-i} \xrightarrow{SRA} R_8,\ Q \xrightarrow{SRA} Q$	16 cycles multiplication $a_i * V_{n-i}$
⋮				⋮	
19				Result in R_7, Q	
20				$R_5 - 1 \rightarrow R_5$	
21				$R_{10} + R_7 \rightarrow R_{10}$	
22				$R_{11} + C_0 \rightarrow R_{11}$	Branch to 2
23				$R_0 - 1 \rightarrow R_0,\ R_0 \rightarrow MA$	Branch to 24 if $R_5 = 0$
24		$R_1 - R_3$		$R_1 - R_3 \rightarrow MA$	
25	S_{n-M}	R_4		$R_4 \rightarrow MA$	
26	α			$S_{n-M} \rightarrow R_8$	
27				$\alpha \xrightarrow{SRA} Q\ S_{n-M} \xrightarrow{SRA} R_8$	16 cycles multiplication $\alpha * S_{n-M}$
⋮				⋮	
42				Result in R_7, Q	
43		R_1		$R_1 \rightarrow MA\ R_1 - 1 \rightarrow R_1$	
44	S_n			$R_{10} + R_7 \rightarrow R_{10}$	
45				$C_0 + R_{11} \rightarrow R_{11}$	
46				$S_n - R_{10} \rightarrow R_{10}$	Branch 1
47				NOP	If $R_{1=0}$ branch exit

$$C_n = S_n - \alpha S_{n-M} - \sum_{i=1}^{4} \alpha_i V_{n-i} \qquad n = 1, 2, \cdots 128$$

Figure 4-9. Second Reduced Signal Inner Loop

pointers. R_5 is used as a loop control.

The inner loop takes 22 x 4 + 23 cycles. The outer loop, which performs the loop control, register initialization and stores the results back in memory requires an additional 12 cycles. Therefore, the total required cycles are:

$(22 \times 4 + 23) \times 128 + 12 \times 128 = 15744$

e. PARCOR

Calculation coefficients of the PARCOR coefficients is a recursive process. For sake of clarity, Figure 4-10 shows only the cycled multiplication part, which contains most of the calculations.

Registers R_0, R_2 are used as memory address pointers. R_6 is used as a loop counter. Double precision accumulation ends up in R_3 and R_4. Each term in the calculation takes two multiplications and each of them cycles 1, 2 and 3 times. In addition, two separate multiplications and four divisions are necessary to complete the task. 30 overhead cycles are added to guarantee enough cycles for memory address indexing, loop control and storing results back into memory, etc.

Therefore, the required cycles are:

$30 \times 3 + 20 \times (1 + 2 + 3) + 20 \times (1 + 2 \times 3) + 132 \times 4 + 20 = 878$

PROGRAM SEQUENCE	Arithmetic Logic Unit				BRANCHING
	A REG	MA REG	O REG	FUNCTION	
1		$R_0 - R_2$		$R_0 - R_2 \rightarrow MA$	
2	A(I-M)			$D \rightarrow R_1$	B(M) = A(I-M)
3		R_0		$R_0 \rightarrow MA$, $R_0 + 1 \rightarrow R_0$	
4	B(M)				
5				$R_1 \xrightarrow{SRA} R_1$, $D \xrightarrow{SRA} Q$	16 cycles multiplication A(I-M)*B(M)
⋮				⋮	
21				Result in R_5, Q	
22				$R_3 + R_5 \rightarrow R_3$	
23				$R_4 + CO \rightarrow R_4$	
24				$R_6 - 1 \rightarrow R_6$	Branch to 1
25				NOP	Branch exit if $R_6 = 0$

Figure 4-10. Partial PARCOR Inner Loop

4.3.3 Timing and Memory Estimates

The overall program timing is derived from Figure 4-3. Total required time for the transmitter side is calculated with some of the calculations taken from the examples shown in the previous section. The result is shown in Table 4-1 below:

FUNCTION	TIME (ms)
Input Normalization	0.71
I/O Interrupt	0.84
AMDF	1.49
α, pitch gain	0.84
Reduced waveform normalization	0.89
Autocorrelation normalization	2.16
PARCOR quantization dequantization	0.13
Predictor coefficients	0.03
Error signal	2.34
Error signal quantization	1.17
Q, quantization level	0.04
Total time	10.62 ms

TABLE 4-1

The Estimates of Transmitter Timing of APCQ

In the APCQ Algorithm, the receiver processing is one-third of the complexity of the transmitter. Therefore, the estimated timing is one-third of that required for the transmitter. Also, allowing 15 percent overhead time for subroutine control, data handling and synchronization, etc., then the overall processing of APCQ algorithm implementation is:

$$10.62 \times (1 + 33\%) \times (1 + 15\%) = 16.2 \text{ msec}$$

Memory used in the LSI processor can be divided into micro and macro memories. Micro memory having a word length of 48 bits is used for micro program storage.

Macro memory having a word length of 16 bits, contains data and the macro program.

The amount of Micro program memory (ROM) is conservatively estimated as shown in Table 4-2:

	Transmitter	Receiver
Input Normalization	110	110
AMDF	30	
α, pitch gain	75	
Division subroutine	55	
Square root subroutine	65	
Quantization subroutine	80	40
Dequantization subroutines	30	30
Reduced waveform and normalization	65	65
Autocorrelation and normalization	60	
Predictor coefficients	50	50
Error signal	60	60
Initialization and synchronization	100	
Q, quantization level	60	
PARCOR	100	
Total	940	355

TABLE 4-2

Microprogram Memory Requirements

Besides the memory required as shown in Table 4-2, we should allow 15% overhead for memory for inputting and outputting the digital data. Therefore, total memory is:

(940 + 355) x (1 + 15%) = 1490

or 1490 x 48 = 71520 bits. This is 6 chips of 16K ROM (1K x 16)* with 500 spare words.

Macro program I/O interrupts, control memory and data buffer for the processor are as shown in Table 4-3.

Data buffer	5 x 128 + 2 x 204
AMDF	100
I/O interrupts + macro program control	350
Total	1498

TABLE 4-3 : Macroprogram Memory Requirements

or 1498 x 16 = 23968 bits. This is 6 chips 4K RAM (512 x 8).*

In addition to these memory chips constant data memory should require 1 chip of 16K ROM (1K x 16)*.

To sum up the above estimation, the estimated memory are:

4K RAM	6 chips
16K ROM	7 chips

* Memories are projected to 1978 technology.

DEVICE I/O	QTY.	POWER	COST
00	5	.180	0.85
02	3	.348	0.51
02	14	1.596	2.52
08	6	0.456	1.02
30	4	0.076	0.68
32	2	0.070	0.46
77	2	0.320	0.46
132	1	0.340	0.59
133	1	0.250	0.17
138	7	1.575	5.56
148	1	0.190	0.58
153	2	0.450	1.28
157	3	0.750	1.92
163	3	1.781	2.37
174	9	4.61	6.75
182	1	0.36	0.68
241	8	5.44	20.56
251	2	0.550	1.28
253	10	.34	5.00
276	7	2.10	3.78
330	1	0.61	10.00
374	4	1.8	10.84
2401	4	3.7	120.00
482	3	1.426	30.00
9900	1	1.000	60.00
4K RAM	6	3	36.00
16K ROM	7	4.2	101.50
12-Bit A/D	1	2.43	195.00
8-Bit D/A	1	1.6	49.00
TOTAL	119	41.748 watts	$669.36

Materials, Quantities, Power and Cost

TABLE 4-4

4.3.4 Cost and Power Estimates

The cost and power estimate of the LSI processor was performed by adding up the cost and power needs of each ch'p. Losses caused by the PC boards, mechanic assembly, power supply, etc., were not included in the estimate, but these are estimated to be small. The parts list, total power and cost are shown in Table 4-4.

4.4 Other LSI Processor Structures

The parallelism of the proposed LSI processor can be further increased by changing the loop processor as shown in Figure 4-11 to include an index processor and hardware multiplier. Now the loop processor presents three functionally independent processors.

The index processor is used for the computation of memory addresses. It consists of a 16-element file of data plus an 8-element array of pointers with a limited ALU.

The hardware multiplier performs 16-bit by 16-bit two's complement multiplication providing 32-bit results in QH and QL registers (16 bits each). Once loaded and started, it will complete its product in 8 cycles (or 4 or 2 depending on design) while other operations can continue in parallel in both the index processor and main ALU.

The main arithmetic/logic unit is the same as we mentioned before. However, its memory addressing function can be performed by the index processor and its multiplication function can be performed by the multiplier processor.

These three sections of the loop processor run synchronously under microprogram control.

In this section, we consider adding either the index processor or the multiplier processor or both, to the main

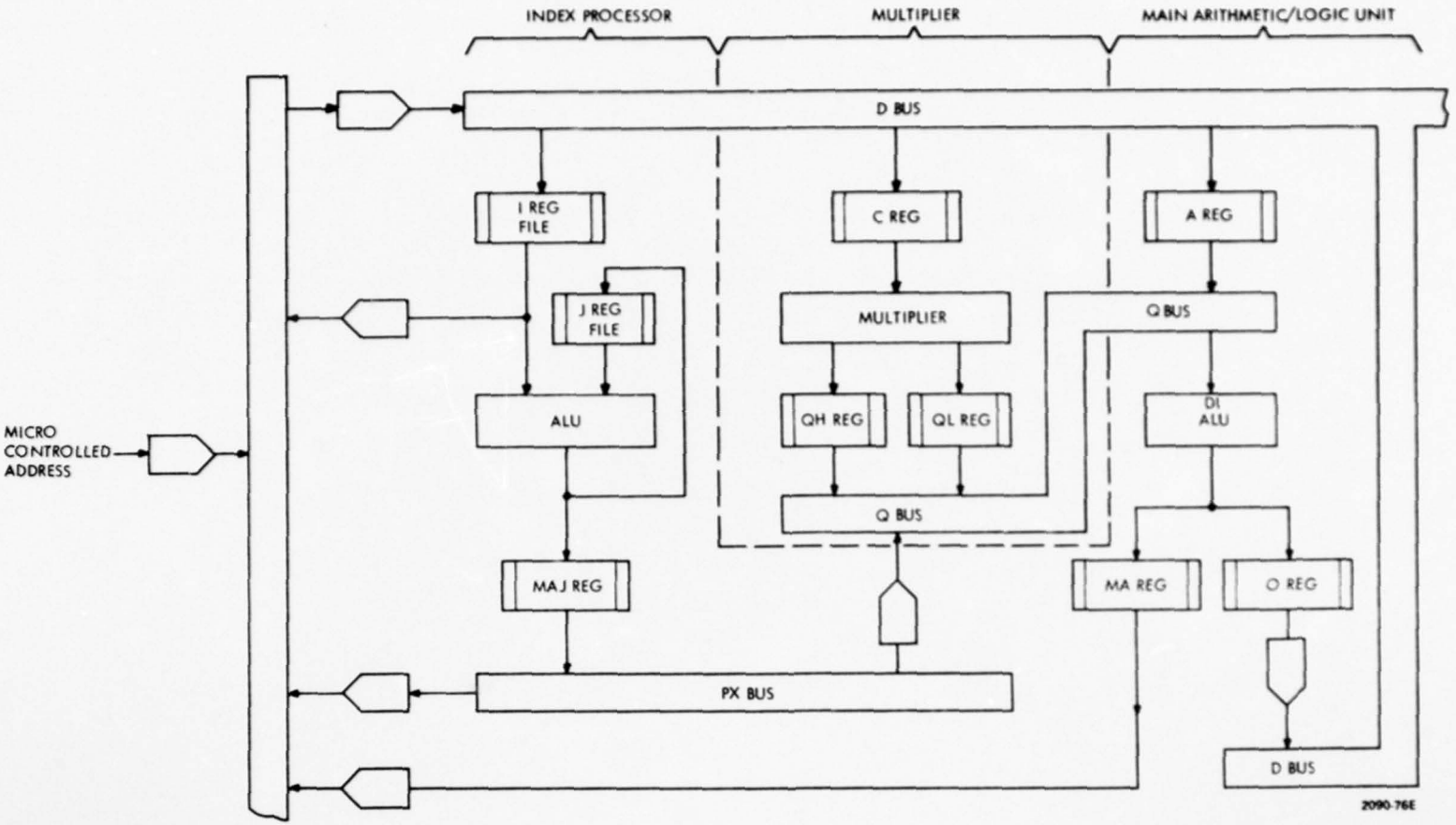


Figure 4-11. Loop Processor with Added Parallelism

ALU. The timing, memory, cost and power will be re-established to compare with the previous single main ALU processor.

4.4.1 Timing and Memory Estimates

With an index processor, the memory address pointer can access memory locations at the same time the main ALU performs other operations. It can, therefore, save one micro program instruction at the beginning of every memory index. With a hardware multiplier, a multiplication requires 10 cycles with an index processor and 11 cycles without an index processor.* Without a hardware multiplier, multiplication requires 20 cycles with an index processor and 22 cycles without an index processor. Hence, the use of a hardware multiplier saves at least half the micro instructions. The added parallelism has increased the complexity of bus structures somewhat and the word length of micro instruction by three bits for the multiplier and 13 bits for the index processor.

The timing of the transmitter side of the APCQ implementation for the different structures is analyzed and listed in Table 4-6.

The receiver processing time is considered to be one-third the required time for the transmitter. Overhead time is 15 percent of the total time required for the transmitter and the receiver.

From Table 4-5, we can conclude that adding a hardware multiplier improves the processing speed by about 30 to 50%. Adding an index processor improves the processor speed by 16 to 25%.

* It is almost always possible to perform additional useful operations in either the ALU or the index processor while the multiplier is performing its function such that the multiply time is effectively reduced.

FUNCTION	TIME (ms)			
	With Index Processors		Without Index Processors	
	With Multiplier	Without Multiplier	With Multiplier	Without Multiplier
Input Normalization	0.28	0.56	0.38	0.71
I/O Interrupt	0.76	0.76	0.84	0.84
AMDF	0.95	0.95	1.49	1.49
α, Pitch gain	0.34	0.66	0.36	0.83
First reduced form & normalization	0.42	0.66	0.42	0.89
Autocorrelation & normalization	0.89	1.77	1.18	2.16
PARCOR & quantization & dequantization	0.07	0.13	0.07	0.13
Predictor coefficients	0.01	0.02	0.01	0.03
Second reduced form (error signal)	0.99	1.77	1.23	2.34
Error signal quantization	0.69	1.08	0.75	1.17
Q, quantization level	0.03	0.03	0.03	0.04
Transmitter processing time	5.55	8.51	6.77	10.62
Receiver processing time	1.85	2.84	2.56	3.54
Overheads	1.11	1.70	1.31	2.12
Total time	8.51	13.05	10.63	16.28

TABLE 4-5

Processor Time Requirements vs. Hardware Complexity

With the different structures micro program memory is different as is the micro instruction word length and memory size.

Table 4-6 shows these differences. Overhead cycles have taken 15% from the total transmitter and receiver memory.

From the table, it is clear that with the indicated index processor added, the overall memory required is increased, because of the increase of micro instruction word length. Other index processors with greater computational capability can provide for greater speed increases with approximately the same increase in micro control memory.

The total memory, which includes both macro and micro memory will be as shown in Table 4-7.

	MEMORY			
	With Index Processor		Without Index Processor	
	With Multiplier	Without Multiplier	With Multiplier	Without Multiplier
Micro instruction Word length	64	61	52	48
Transmitter Memory size	605	780	730	940
Receiver Memory size	205	265	275	355
Overheads	122	157	151	195
Total Bits	59648	73322	60112	71520
Required 16K ROM chips	4	8 (with 800 spare words)	8 (with 800 spare words)	6 (with 500 spare words)

TABLE 4-6

Memory Requirements vs. Hardware Complexity

	MEMORY			
	With Index Processor		Without Index Processor	
	With Multiplier	Without Multiplier	With Multiplier	Without Multiplier
16K ROM for micro-memory	4	8	8	6
4K RAM for macro-memory	6	6	6	6
16K ROM for macro-memory	1	1	1	1

TABLE 4-7 Number of Memory Chips

Adjustment	With Index Processor				Without Index Processor			
	With Multiplier		Without Multiplier		With Multiplier		Without Multiplier	
	Power	Cost	Power	Cost	Power	Cost	Power	Cost
Multiplier	+2.5	50			2.5	50		
Index Processor	5.11	86	5.11	86				
Memory	-1.2	-29	+1.2	+29	1.2	+29		
Base	41.158	669.36	41.748	669.36	41.748	669.36	41.748	669.36
Total	48.158 watts	$776.36	48.058 watts	$784.36	45.448 watts	$748.36	41.748 watts	$669.36

TABLE 4-8 Cost and Power of Various Forms of LSI Processors

4.4.2 Estimates of Cost and Power of Index Processor and Multiplier

The cost and power required for the index processor is $85 and 5.11 watts respectively. For the multiplier they are $50 and 2.5 watts respectively.

Therefore, the cost and power for the modified processor appear in Table 4-8.

4.5 Discussions

Table 4-5 and Table 4-8 have shown us many significant results. The cost and power do not differ greatly for the different structures, but the speed does. The best choice for APCQ seems to be the processor with multiplier, but without index processor.

SECTION 5

RECOMMENDATIONS AND CONCLUSIONS

5.1 Conclusions

This contract has resulted in the development of a high quality 16 Kb/s speech digitizer based on adaptive predictive coding principles which is robust to channel errors and relatively simple to implement. Informal listening tests indicate that the speech quality of the 3000 Hz audio bandwidth adaptive predictive coder is noticeably superior to Continuous Variable Slope Deltamodulation (CVSD) which we also developed as a benchmark. Experimentation with the CVSD algorithms indicated they could be improved slightly by adjusting internal parameters, especially the compression ratio defined as the maximum quantizer level over the minimum level.

Although this change lessened the slope overload distortion, making the speech sound less muffled, its voice quality was always lower than that of the Adaptive Predictive coder at 16 Kbps.

Using commercially available LSI electronics the 16 Kbps adaptive predictive coder could be implemented using about $700 in parts and would consume about 42 watts of power. Our engineering estimates of the hardware incorporated the Advanced Microprocessor Devices AMD2901 microprocessor chip to develop a programmable multilevel adaptive predictive coder.

This approach, while more complex electronically than that used for CVSD, would result in a complete telephone terminal with built in telephone line synchronization, modem interface and key interface at about the same cost as a terminal employing a CVSD speech digitizer. This occurs because the programmable approach

can timeshare the microprocessor hardware among the speech digitizer and these other tasks while the terminal with the CVSD digitizer must have special hardware for each task. Because they must have this specialized hardware, present CVSD terminals are surprisingly costly, even though the CVSD digitizer is extremely simple. Thus, for about the same terminal cost, the multilevel adaptive predictive coders developed under this contract would noticeably improve the voice quality over the CVSD. Consequently, these coders should be studied and refined still further.

5.2 Recommendations

The final multilevel adaptive predictive coder system determined under this contract had a 3100 Hz bandwidth and a 5 level fixed/frame quantizer. Furthermore, it employed both a pitch loop predictor and a coefficient predictor. The fixed/frame quantizer was used because, at the design review, it was the only quantizer known that was robust to channel errors. The 5 level quantizer was used because it was the most straightforward approach to obtaining a 3000 Hz audio bandwidth and the pitch loop was necessary because the speech quality degraded when it was removed.

Just as the contract ended, we started to investigate modified Jayant quantizers and other quantizers which are also robust to channel errors. Preliminary investigations indicate that these quantizers provide improved quality over the fixed/frame quantizer, and their use can eliminate the need for the pitch loop without loss of quality. We recommend that these robust quantizers be studied further, and that they be incorporated into PSP software for operation in the DCA equipment. This will provide high quality adaptive predictive coders using these quantizers. In addition, we recommend

that variable length coding coupled with storage buffers be used to obtain a 3000 Hz audio bandwidth with either a 7 or 8 level quantizer. This will improve the signal-to-noise ratio of the system while keeping the transmission rate at 16 Kb/s. These studies should simplify the implementation hardware from the system being delivered under this contract, while improving voice quality still further.

REFERENCES

1. A. J. Goldberg, A. Shaffer, "A Real Time Adaptive Predictive Coder Using Small Computers," IEEE Trans. Communications, Vol.COM-23, No. 12, Dec. 1975.

2. B. S. Atal and M. R. Schroeder, "Adaptive Predictive Coding of Speech Signals," Bell Syst. Tech. J., pp. 1973-1986, Oct. 1970.

3. J. M. Kelly, et al., "Final Report on Predictive Coding of Speech Signals," Bell Laboratories, rep. on Army Contract DAABO3-69-C-0338, June 1970.

4. N. S. Jayant, "Digital Coding of Speech Waveforms: PCM, DPCM, and DM Quantizers," Proc. IEEE, Vol. 62, No. 5, pp. 611-632, May 1974.

5. S. U. H. Qureshi and G. D. Forney, "Adaptive Residual Coder - An Experimental 9.6/16Kb/s Speech Digitizer, EASCON, 1975 Conf. Rec., pp. 29A-29E, Sept. 1975.

6. M. J. Ross, H. L. Shaffer, A. Cohen, R. Freudberg, and H. J. Manley, "Average Magnitude Difference Function Pitch Extractor," IEEE Trans. Acoust., Speech, Signal Processing, vol. ASSP-22, pp. 353-362, Oct. 1974.

7. J. D. Markel and A. H. Grey, Jr., "On Autocorrelation Equations as Applied to Speech Analysis," IEEE Trans. Audio Electroacoust., vol. AU-21, pp. 69-79, Apr. 1973.

8. B. S. Atal and S. L. Hanauer, "Speech Analysis and Synthesis by Linear Prediction of the Speech Wave," J. Acoust. Soc. Amer., vol. 50, pp. 637-655, Aug. 1971.

9. P. Noll, "The Presence of PCM and DPCM Speech Coders in the Presence of Independent and Correlated Errors," ICC Conference Record, pp. 30-1 through 30-5, June 1975.

10. P. Noll, "A Comparative Study of Various Quantization Schemes for Speech Encoding," Bell System Technical Journal, Vol. 54, No. 9, pp. 1597-1614, Nov. 1975.

11. D. J. Goodman and R. M. Wilkinson, "A Robust Quantizer," IEEE Trans. on Communications, Vol. COM-23, No. 11, Nov. 1975, pp. 1362-1364.

APPENDIX A
MORE DETAILED FLOW CHARTS OF APCQ CODER SIMULATION

A.1 INTRODUCTION

Figures A-1 and A-2 contain more detailed flow charts of the fixed-point simulation with 8 level Jayant Quantizer. These characters indicate the scaling needed to process the speech with 16-bit arithmetic. This scaling is necessary since the summing of the data can cause the answer to overflow and exceed the maximum number of the machine (2^{15}). Additionally, scaling is needed to avoid underflow in multiplication since only 16 bits of a 32-bit product of two 16-bit numbers can be retained. It is common practice to retain the upper or most significant 16 bits, which is equivalent to dividing the 32-bit product by 2^{15}. This operation is performed automatically by the hardware and is called a fractional multiply. Here the numbers are thought of as fractions having a magnitude less than 1, and the location of the binary point is not changed after multiplication. The flow charts indicate this fractional multiply by showing the division by 2^{15}, which, as previously stated, is performed automatically by the hardware.

A.2 MAIN PROGRAM FLOW CHARTS AND LISTINGS

Figure A-1 presents a flow chart of the analyzer section of the APCQ Coder. In this portion of the program, the data is read off the tape, pre-emphasized, normalized, and processed to compute the pitch M, pitch gain ALPHA. This portion of the program also includes computations and quantizations of the residual error signal and the encoding of the quantized error signal. Additionally, the PARCOR parameters are calculated by this code.

Figure A-2 similarly describes the receiver portion of the code. Here the transmission parameters are decoded, the predictor coefficients recomputed from the PARCOR coefficient and the error signal used to excite the reconstruction filter. Prior to output on tape, the speech is de-normalized and de-emphasized.

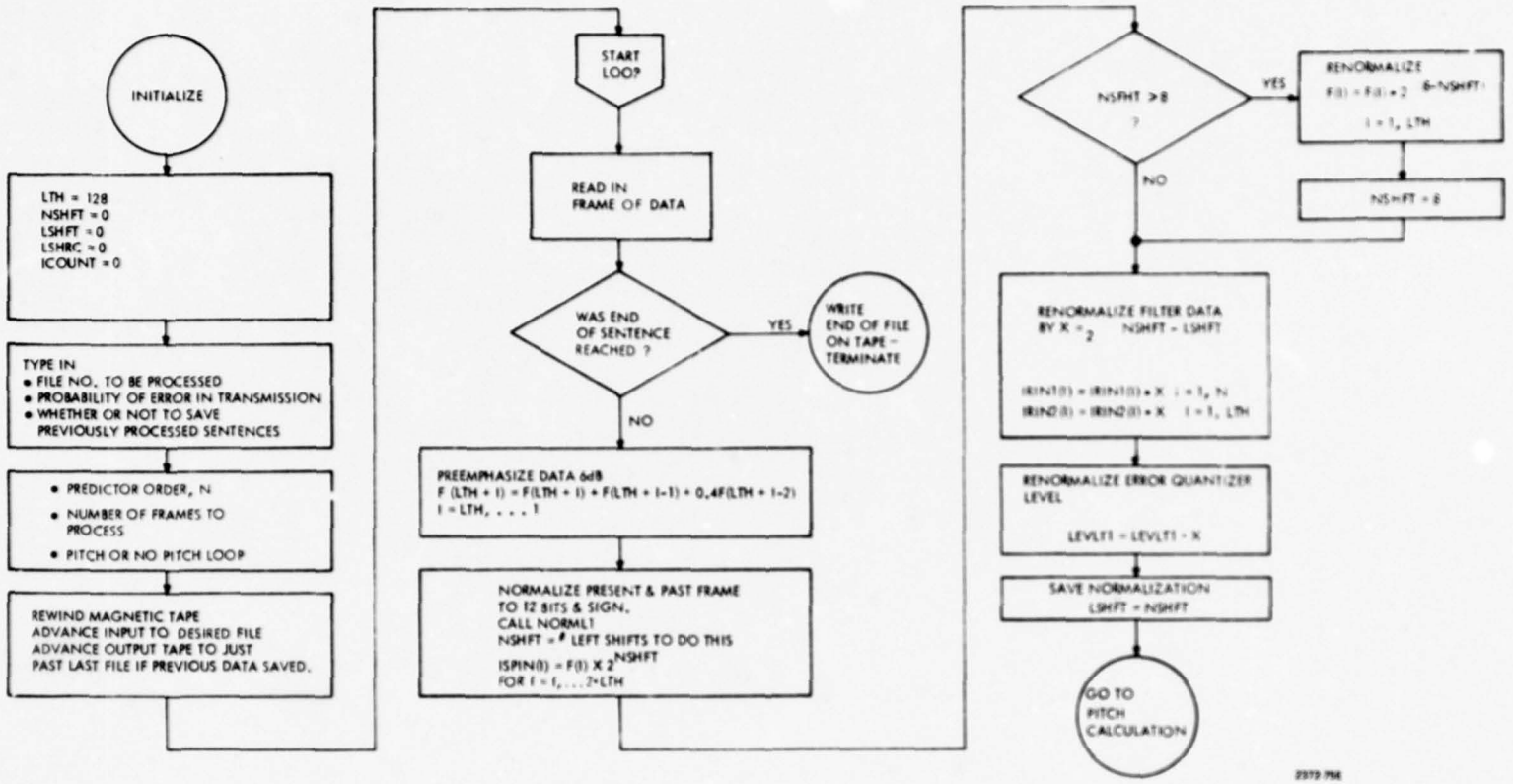


Figure A-1. Flow Chart of Fixed-Point APCQ Analyzer (Sheet 1 of 3)

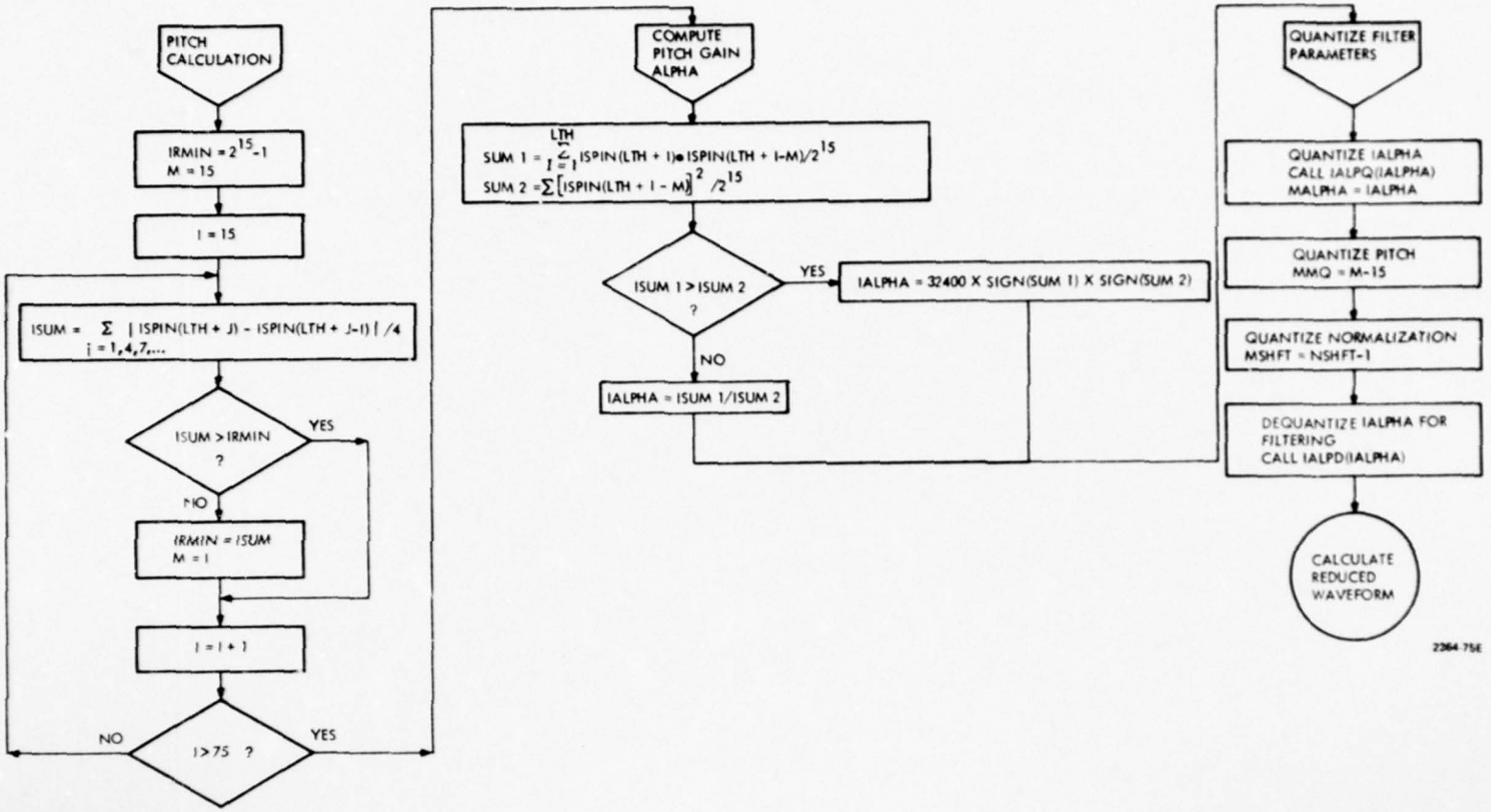


Figure A-1. Flow Chart of Fixed-Point APCQ Coder-Analyzer (Sheet 2 of 3)

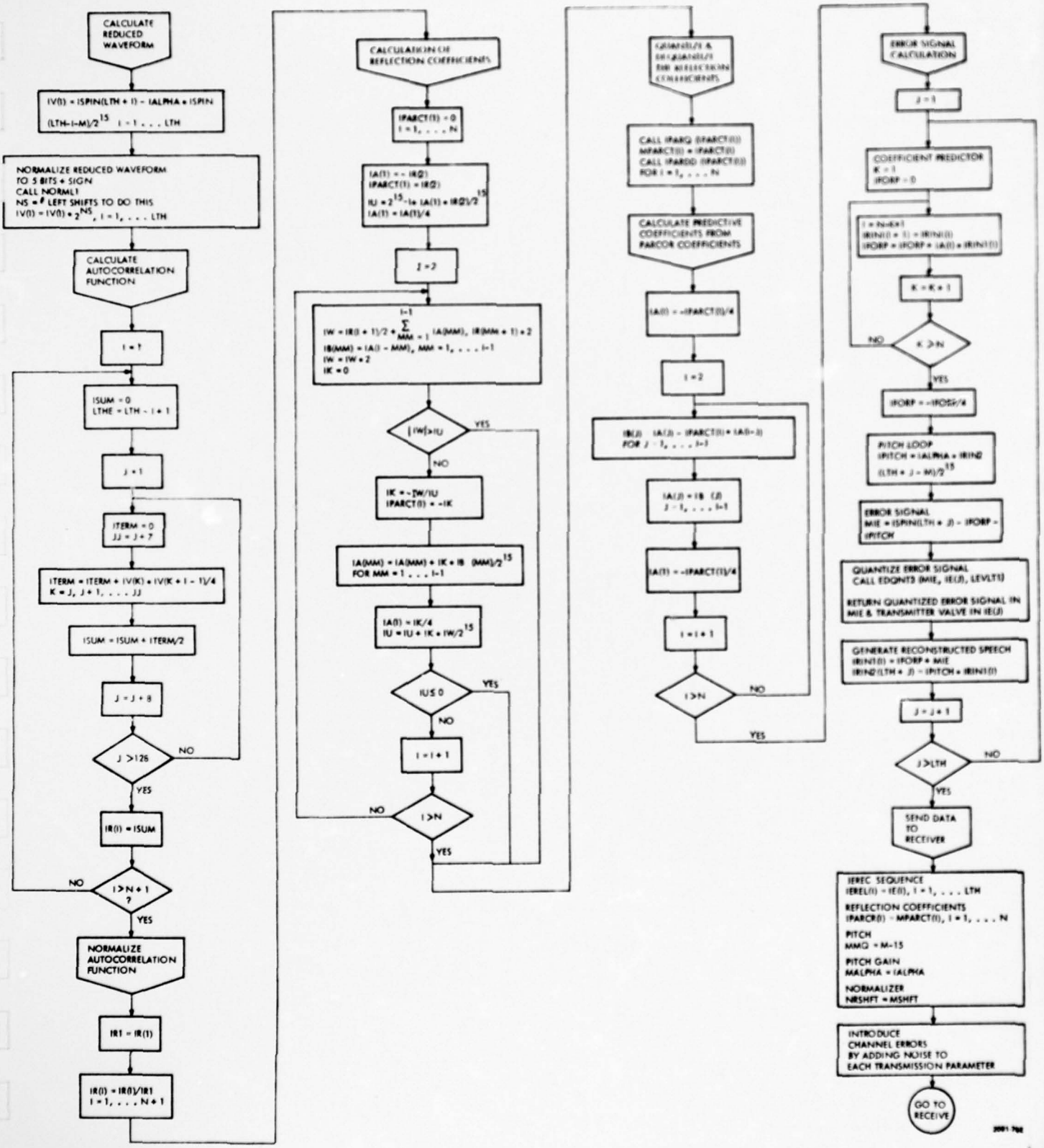


Figure A-1. Flow Chart of Fixed-Point APCQ Coder-Analyzer (Sheet 3 of 3)

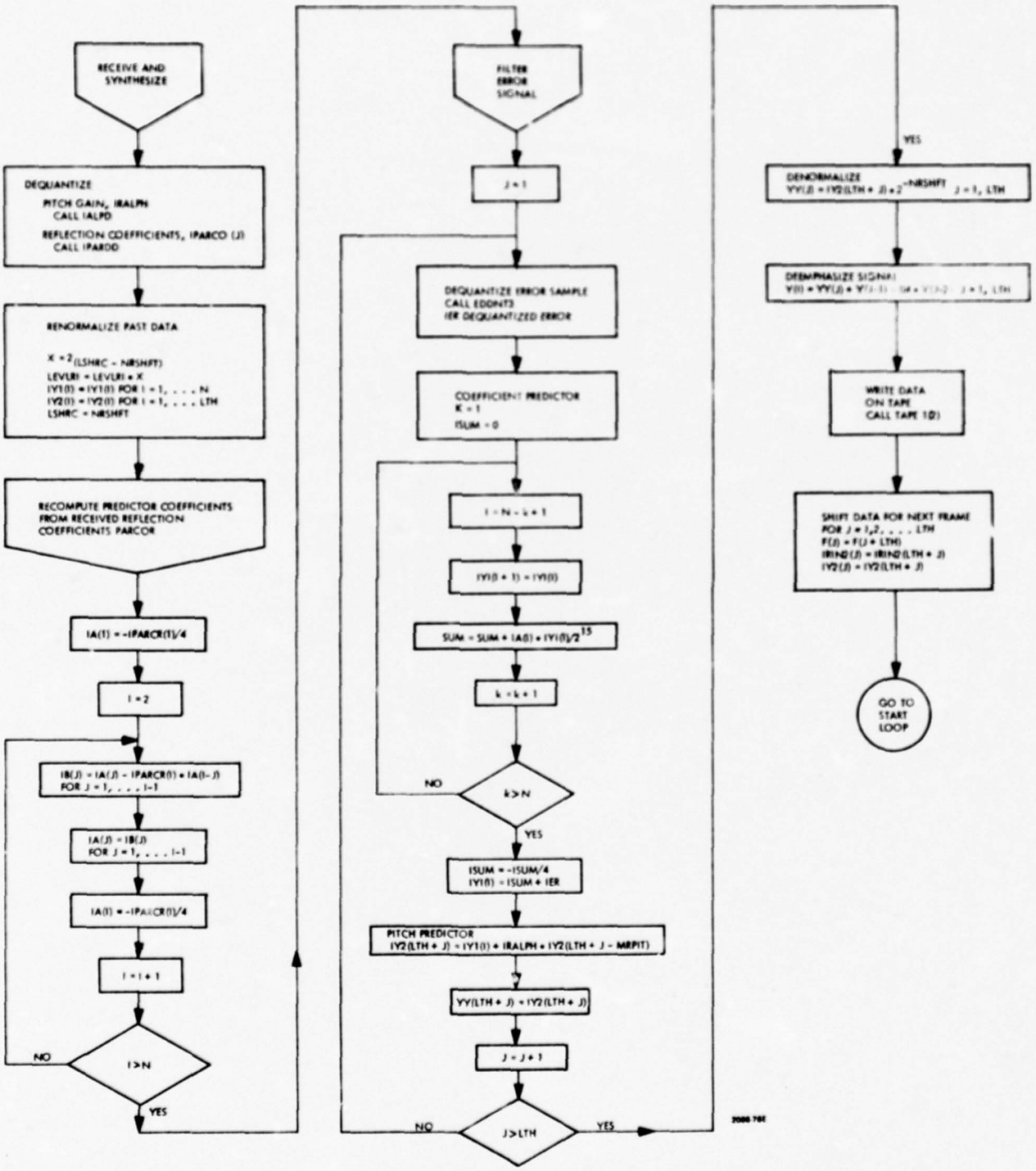


Figure A-2. Flow Chart of Fixed-Point APCQ Synthesizer

Table A-1 contains a list of the variable names and their meanings as used in the main program of the APCQ Coder. Included in this Table is a list of the subroutines and arithmetic functions used as well as the data arrays needed to filter the speech data.

The FORTRAN listing of the main program is contained in Figure A-3 and includes comment statements indicating the main functions of the program.

TABLE A-1. VARIABLE NAMES USED IN MAIN PROGRAM OF APCQ CODER

SUBROUTINES	MEANING
TAPE1(I)	Magnetic tape handling where I indicates function: I=1 read I=2 write I=3 rewind tapes I=4 search for last file I=5 write end of file I=6 rewind only input-search for desired input file
SW1(I, ITOG)	Set ITOG=1 if console switch I is up, ITOG=0 if down
DISREG (I)	Display I on console
NORML1 (V,VV,L,N,NS)	Normalize N numbers in array V to N+1 bits & place result in VV. NS=# shifts to do this
RANDIN (I,J,Y)	Random number generator
IALPQ (IALPHA)	Quantizes pitch gain to 3 bits
IALPD (IALPHA)	Dequantize IALPHA
EDQNT3(MIE,IE(J), LEVLT1)	Code error vector MIE into transmission vector IE(J)
RANERR (A1,A2,A3,A4,A5,A6)	Inserts random errors into data stream A1 according to A2-A6
EDDNT3(IEREC(J), IER, LEVLR1)	Dequantize error IEREC(J) into IER, adjust Level LEVLR1
IPARQ(IPARCT(J))	Quantize PARCOR (Reflection coefficient parameter IPAR CT(J)
IPAROD(IPARCT(J))	Dequantize PARCOR transmission parameter

```
C       PROGRAM DCA#APC.FTN
C       THIS PROGRAM SIMULATES THE APCQ SYSTEM
C       WITH FIXED POINT ARITHMETIC
C       AND WITH ADAPTIVE PREDICTION AND PITCH LOOP
C*****
C******
C******

        COMMON/MTAPE0/NIN(170),NOUT(170)
        COMMON/MTAPE1/NSKIP,IST,NTOTI,NTUPS,NTOTO
        COMMON/MTAPE2/NEND,NERR,NFILE
        DIMENSION ISPIN(260)
        DIMENSION IR(10),IV(150),IE(130),IA(10),IPARCR(10)
        DIMENSION MPARCT(10),IPARCT(10)
        DIMENSION F(260),Y(130)
        DIMENSION YY(130),IY2(260),IRIN2(260)
        DIMENSION IRIN1(10),IEREC(130),IB(10),IY1(10)
C       INITIALIZATION OF QUANTIZER
1010    CONTINUE
C
        LEVLR1=1
        LEVLT1=1
        LP=1024
        WRITE(6,1177)
        F215=2.**15
        FL215D=F215-1.
C***
C       TYPE IN INFORMATION ABOUT FILE TO BE PROCESSED
C***
1177    FORMAT(1X,'FILE NO. ='/)
        READ (6,101)NFILE
101     FORMAT(I3)
        NTOTI=128
        NTUPS=NTOTI
        NTOTO=NTOTI
        LTH=NTOTI
        LTH2=LTH*2
        LTHH=75
        IST=1
        JRN=0
        IRN=0
        NSHFT=0
        LSHFT=0
        LSHRC=0
        CSN=0.
C***
C       PREDICTOR ORDER
C***
        WRITE(6,99)
99      FORMAT(1X,'PREDICTOR ORDER='/)
        READ (6,101)N
        NN=N+1
C***
C       PROCESS FROM BEGINNING OF FILE
C***
        NSKIP=1
        NSKIPS=NSKIP
C**
C       NUMBER OF PROCESSING FRAMES
C***
        IREC=999
        WRITE(6,105)
```

```
C***
C       ASK IF WE WISH TO APPEND THIS DATA TO MAGNETIC TAPE
C       AND SAVE ALL PREVIOUS PROCESSING ON TAPE
C***
105     FORMAT(1X,'1=APPEND OUTPUT, 0= BEG TAPE'/)
        READ (6,106)IN
106     FORMAT(I1)
C***
C       ASK ABOUT THE PROBABILITY OF ERROR ON TRNASMISSION
C       PARAMETERS FOR THE ERROR SIGNAL , THE NORMALIZER
C       THE PARCOR COEFFICIENTS, PITCH AND PITCH GAIN
C***
        WRITE(6,208)
208     FORMAT(' FRACTION OF ERRORS IN ERROR SIGNAL'/)
        READ(6,209) PROBA
        WRITE(6,1208)
1208    FORMAT(' FRACTION OF  ERRORS IN THE OTHER PARAMETERS'/)
        READ(6,209) PROB
209     FORMAT(F10.0)
C***
C       IS THERE PITCH ON THIS RUN
C***
        WRITE(6,210)
210     FORMAT(1X,'1=PITCH, 0=NO PITCH'/)
        READ (6,106) IPTN
        M=15
        MMQ=0
C***
C       INITIALIZE RANDOM NUMBER GENERATOR
C***
        DO 1216 J=1,49
1216    CALL RANDU (IRN,JRN,YQR)
C***
C       REWIND THE TAPE
C***
        CALL TAPE1(3)
        IF(IN.EQ.0) GO TO 107
C***
C       ADVANCE THE TAPE TO THE END OF THE WRITTEN MATERIAL
C       IF PREVIOUSLY REQUESTED
C***
        CALL TAPE1 (4)
        IF(NERR.EQ.0) GO TO 107
        WRITE(6,109)
109     FORMAT(1X,'FILE NOT FOUND'/)
        GO TO 1010
C***
107     LTHX=LTH-2
C***
C       READ FRONT CONSOLE SWITCH REGISTER
C***
        CALL SW1(15,ITOG)
        ICOUNT=0
        JCOUNT=0
C***
C       SW 15 CHANGES - CAUSES END OF RUN
C       SW 13 UP - PRINT CHANNEL ERROR INFORMATION
C***
1000    CALL SW1(15,ITOG1)
        CALL SW1(13,JQS)
        IF(ITOG1.NE.ITOG) GO TO 5000
```

```
      ICOUNT=ICOUNT+1
      IF(ICOUNT.GT.IREC) GO TO 5000
C***
C     DISPLAY NUMBER OF PROCESSING FRAME ON COMPUTER CONSOLE
C***
      CALL DISREG(ICOUNT)
C*****
C     DATA INPUT
C*****
      CALL TAPE1(1)
      IF(NEND.NE.0) GO TO 5000
      DO 1001 J=1,LTH
1001  F(LTH+J)=NIN(J)
      FN1=F(LTH2)
      FN2=F(LTH2-1)
C***
C     PREEMPHAISZE SPEECH 6 DB. ABOVE 500 HZ.
      DO 1101 J=1,LTHX
      I=LTH2-J+1
1101  F(I)=F(I)-F(I-1)+0.4*F(I-2)
      F(LTH+2)=F(LTH+2)-F(LTH+1)+0.4*FL1
      F(LTH+1)=F(LTH+1)-FL1+0.4*FL2
      FL1=FN1
      FL2=FN2
C***
C     NORMALIZE PRESENT AND PAST ANALYSIS INTERVALS
C*****
      DO 1005 J=1,LTH2
1005  ISPIN(J)=F(J)
C****
C     NORMALIZE PRESENT AND PAST FRAMES TO 12 BITS + SIGN
C****
      CALL NORML1(ISPIN,ISPIN,LTH2,11,NSHFT)
C     DO NOT SHIFT THE DATA MORE THAN 8 PLACES AND
C     ALWAYS SHIFT IT AT LEAST 1 PLACE
      IF(NSHFT.LE.8) GO TO 2600
      NP=2**(NSHFT-8)
      DO 2601 I=1,LTH2
2601  ISPIN(I)=ISPIN(I)/NP
      NSHFT=8
C  SET UP NSHFT FOR TRANSMISSION.  NSHFT QUANTIZED TO 4 BITS.
2600  NSHFT=NSHFT-1
C****
C     MULTIPLY DATA IN ANALYZER BY 2**(NSHFT-LSHFT)
C****
      IF(NSHFT-LSHFT)1030,1031,1032
1030  NL=2**(LSHFT-NSHFT)
      DO 1108 I=1,N
1108  IRIN1(I)=IRIN1(I)/NL
      DO 1007 I=1,LTH
1007  IRIN2(I)=IRIN2(I)/NL
      LEVLT1=LEVLT1/NL
      IF(LEVLT1.EQ.0) LEVLT1=1
      GO TO 1031
1032  NL=2**(NSHFT-LSHFT)
      DO 1109 I=1,N
1109  IRIN1(I)=IRIN1(I)*NL
      DO 1006 I=1,LTH
1006  IRIN2(I)=IRIN2(I)*NL
      LEVLT1=LEVLT1*NL
      IF(LEVLT1.GE.LP)LEVLT1=LP
```

```
1031    LSHFT=NSHFT
C
C       CALC OF AMDF AND SELECTION OF M
C
        IF(IPTN.EQ.0) GO TO 700
        M=15
        IRMIN=FL215D
        DO 2 I=15,75
        ISUM=0
        DO 1 J=1,LTHH,3
        JJ=LTH+J
1       ISUM=ISUM+(IABS(ISPIN(JJ)-ISPIN(JJ-I)))/4
        IF(ISUM.GT.IRMIN) GOTO 2
        M=I
        IRMIN=ISUM
2       CONTINUE
C  SET UP M FOR TRANSMISSION.  M QUANTIZED TO 6 BITS.
        MMQ=M-15
C*****
C       COMPUTES ALPHA
C*****
        ISUM1=0
        ISUM2=0
        DO 40 J=1,LTH
        JJ=LTH+J
        ISP=ISPIN(JJ-M)
        ISUM1=ISUM1+IFMUL(ISPIN(JJ),ISP)
40      ISUM2=ISUM2+IFMUL(ISP,ISP)
        IALPHA=0
        IF(ISUM2.EQ.0) GO TO 700
        IF(IABS(ISUM1).GE.IABS(ISUM2)) GO TO 702
        IALPHA=IFDIV(ISUM1,ISUM2)
        GO TO 700
702     IALPHA=32400*(ISUM1/IABS(ISUM1))*(ISUM2/IABS(ISUM2))
C***
C***
700     CALL IALPQ(IALPHA)
        MALPHA=IALPHA
        CALL IALPD(IALPHA)
C****
C       CALCULATE REDUCED WAVEFORM
C***
        DO 41 I=1,LTH
        II=LTH+I
41      IV(I)=ISPIN(II)-IFMUL(IALPHA,ISPIN(II-M))
C***
C       INTEGER CALCULATION OF AUTOCORRELATION FUNCTIONC
C***
        CALL NORML1(IV,IV,LTH,5,NS)
        DO 5 I=1,NN
        ISUM=0
        LTHE=LTH-I+1
        DO 6 J=1,LTHE,8
        ITERM=0
        JJ=J+7
        DO 9 K=J,JJ
9       ITERM=ITERM+IV(K)*IV(K+I-1)/4
6       ISUM=ISUM+ITERM/2
5       IR(I)=ISUM
        IR1=IR(1)
        DO 7 I=1,NN
```

```
7       IR(I)=IFDIV(IR(I),IR1)
C***
C       CALCULATION OF REFLECTION COEFFICIENTS
C***
        DO 3110 I=1,N
3110    IPARCT(I)=0
        IA(1)=-IR(2)
        IPARCT(1)=IR(2)
        IU=FL215D
        IU=IU+IFMUL(IA(1),IR(2))
        IA(1)=IA(1)/4
        IF(N.LE.1) GO TO 31
        DO 30 I=2,N
        IW=IR(I+1)/2
        IZ=I-1
        DO 10 MM=1,IZ
        IB(MM)=IA(I-MM)
10      IW=IW+IFMUL(IB(MM),IR(MM+1))*2
        IW=IW*2
        IK=0
        IF(IABS(IW).GE.IU) GO TO 31
        IK=-IFDIV(IW,IU)
        IPARCT(I)=-IK
15      DO 20 MM=1,IZ
20      IA(MM)=IA(MM)+IFMUL(IK,IB(MM))
        IA(I)=IK/4
        IU=IU+IFMUL(IK,IW)
        IF(IU.LE.0) GO TO 31
30      CONTINUE
C***
C       QUANTIZATION OF THE REFLECTION COEFFICIENTS
C***
31      IF(N.EQ.0) GO TO 4802
        DO 600 I=1,N
        CALL IPAR0(IPARCT(I))
C***
        MPARCT(I)=IPARCT(I)
600     CALL IPARDD(IPARCT(I))
C***
C       CALCULATE PREDICTOR COEFFICIENTS FROM PARCOR COEFFICIENTS
C***
        IA(1)=-IPARCT(1)/4
        DO 120 I=2,N
        IM1=I-1
        DO 110 J=1,IM1
110     IB(J)=IA(J)-IFMUL(IPARCT(I),IA(I-J))
        DO 112 J=1,IM1
112     IA(J)=IB(J)
120     IA(I)=-IPARCT(I)/4
C***
C       CALCULATE ERROR SIGNAL
C***
4802    DO 3100 J=1,LTH
        JJ=LTH+J
C***
C       ERROR INDICATOR
C***
        ICNT=0
        IFORP=0
        IF(N.EQ.0) GO TO 4303
        DO 3320 K=1,N
```

```
      I=N-K+1
      IRIN1(I+1)=IRIN1(I)
3320  IFORP=IFORP+IFMUL(IA(I),IRIN1(I))
      IFORP=-IFORP*4
4803  IPITCH=IFMUL(IALPHA,IRIN2(JJ-M))
      MIE=ISPIN(JJ)-IFORP-IPITCH
C***
C     QUANTIZE ERROR SIGNAL AND CODE ERROR SIGNAL
C***
      CALL EDQNT3(MIE,IE(J),LEVLT1)
      IRIN1(1)=IFORP+MIE
3100  IRIN2(JJ)=IPITCH+IRIN1(1)
C***
C     TRANSFER THE QUANTIZED ERROR SIGNAL
C***
      DO 7540 I=1,LTH
7540  IEREC(I)=IE(I)
C***
C     TRANSFER QUANTIZER PARCOR
C***
      IF(N.EQ.0) GO TO 4804
      DO 7550 I=1,N
7550  IPARCR(I)=MPARCT(I)
C****
C     INTRODUCE THE CHANNEL ERRORS
C***
C  RANDOM NO. IS LESS THAN OR EQUAL TO PROB, AN ERROR IS INTRODUCED
C  AT THAT BIT.
C***
4804  KOUNT=0
      IF(IPTN.EQ.0) GO TO 4806
      IF(PROB.EQ.0.) GO TO 7706
      DO 7701J=1,6
7701  CALL RANERR(MMQ,J,PROB,IRN,JRN,KOUNT)
      DO 7703 J=1,4
7703  CALL RANERR(MALPHA,J,PROB,IRN,JRN,KOUNT)
4806  IF(N.EQ.0) GO TO 4805
      IPL=4
      DO 7704 I=1,N
      DO 7704 J=1,IPL
7704  CALL RANERR(IPARCR(I),J,PROB,IRN,JRN,KOUNT)
4805  DO 7705 J=1,3
7705  CALL RANERR(MSHFT,J,PROB,IRN,JRN,KOUNT)
7706  IF(PROBA.EQ.0.) GO TO 7710
      DO 7707 I=1,LTH
      DO 7707 J=1,3
7707  CALL RANERR(IEREC(I),J,PROBA,IRN,JRN,KOUNT)
      WRITE(5,7711) KOUNT,ICOUNT
7711  FORMAT(' ERRORS',215)
C***
C     RECEIVE
C***
7710  MRPIT=MMQ+15
      NRSHFT=MSHFT+1
      IRALPH=MALPHA
      CALL IALPD(IRALPH)
      IF(N.EQ.0) GO TO 4807
      DO 7220 J=1,N
7220  CALL IPARDD(IPARCR(J))
C***
C*****
```

```
C       SYNTHESIZE OUTPUT
C***
C       MULTIPLY PREVIOUS FRAME BY 2**(NRSHFT-LSHRC)
C***
4807    IACNT=0
        IF(NRSHFT-LSHRC) 5065,5067,5067
5065    NL=2**(LSHRC-NRSHFT)
        LEVLR1=LEVLR1/NL
        IF(LEVLR1.EQ.0) LEVLR1=1
        DO 5168 I=1,N
5168    IY1(I)=IY1(I)/NL
        DO 5068 I=1,LTH
5068    IY2(I)=IY2(I)/NL
        GO TO 5070
5067    NL=2**(NRSHFT-LSHRC)
        DO 5169 I=1,N
5169    IY1(I)=IIMUL(IY1(I),NL,IACNT)
        DO 5069 I=1,LTH
5069    IY2(I)=IIMUL(IY2(I),NL,IACNT)
        LEVLR1=IIMUL(LEVLR1,NL,IACNT)
        IF(LEVLR1.GE.LP)LEVLR1=LP
5070    LSHRC=NRSHFT
C***
C       RECOMPUTE THE PREDICTOR COEFFICIENTS
C***
        IF(N.EQ.0) GO TO 7850
        IA(1)=-IPARCR(1)/4
        IF(N.EQ.1) GO TO 7850
        DO 7810 I=2,N
        IM1=I-1
        DO 7820 J=1,IM1
7820    IB(J)=IA(J)-IFMUL(IPARCR(I),IA(I-J))
        DO 7830 J=1,IM1
7830    IA(J)=IB(J)
7810    IA(I)=-IPARCR(I)/4
C***
C       FILTER THE ERROR SIGNAL
C***
7850    DO 401 J=1,LTH
        JJ=LTH+J
C***
C       DEQUANTIZE ERROR SAMPLE
C***
        CALL EDDNT3(IEREC(J),IER,LEVLR1)
        ISUM=0
        IF(N.EQ.0) GO TO 4810
        DO 400 K=1,N
        I=N-K+1
        IY1(I+1)=IY1(I)
        ITERM=IFMUL(IA(I),IY1(I))
400     ISUM=IADD(ISUM,ITERM,IACNT)
        ISUM=IIMUL(ISUM,-4,IACNT)
4810    IY1(1)=IADD(ISUM,IER,IACNT)
        ITERM=IFMUL(IRALPH,IY2(JJ-MRPIT))
        IY2(JJ)=IADD(IY1(1),ITERM,IACNT)
401     YY(J)=IY2(JJ)
C***
C       END OF SYNTHESIS
C***
C***
C       MULTIPLY OUTPUT BY 2**(NRSHFT)
```

```
C***
4013    XNL=2.**(-NRSHFT)
        DO 4010 J=1,LTH
4010    YY(J)=YY(J)*XNL
C***
C       COMPUTE S/N RATIO
C***
        SUM1=0.0
        SUM2=0.0
        DO 456 I=1,LTH
        FX=F(LTH+I)
        SUM1=SUM1+(FX-YY(I))**2
456     SUM2=SUM2+FX*FX
        SN=10.*ALOG10(SUM2/SUM1)
        CSN=(ICOUNT/(1.+ICOUNT))*CSN+SN/(1.+ICOUNT)
        WRITE(5,666)ICOUNT,SN,CSN
666     FORMAT(1X,'FRAME=',I3,3X,'S/N=',F12.4,3X,'CSN=',F12.4)
        IF(ICNT.EQ.0) GO TO 7385
        WRITE(5,7386)
7386    FORMAT(1X,'ICNT NE 0'/)
        GO TO 7370
7385    IF( ICT.EQ.0) GO TO 7382
        WRITE(5,7390)
7390    FORMAT(1X,'ICT NE 0'/)
        GO TO 7370
7382    IF(IACNT.EQ.0) GO TO 7371
        WRITE(5,7391)
7391    FORMAT(1X,'IACNT NE 0'/)
        GO TO 7370
7371    CONTINUE
        IF (JQS .EQ. 0) GO TO 7380
C  WRITE OUT PARAMETERS IF SWITCH 13 IS UP.
7370    WRITE(5,7901)ICOUNT
        WRITE(5,7911) M,IALPHA,NSHFT,(IPARCT(I),I=1,N),(IE(I),I=1,LTH)
        WRITE(5,7907)
        WRITE(5,7912)MRPIT,IRALPH,NRSHFT,(IPARCR(I),I=1,N),(IEREC(I),I=1,LTH)
        WRITE(5,7913)
7380    CONTINUE
7901    FORMAT(/' ORIGINAL TRANSMISSION PARAMETERS',I5)
7903    FORMAT(/' QUANTIZED PARAMETERS')
7905    FORMAT(/' QUANTIZED PARAMETERS WITH ERRORS')
7907    FORMAT(/' PARAMETERS RECEIVED')
7911    FORMAT(3X,'M=',I3,4X,'IALPHA=',I6,4X,
     1 'NSHFT=',I3/3X,'IPARCT',4(3X,I7)/
        2  3X,'IE ',9I6/13(10I6/)/)
7912    FORMAT(3X,'MRPIT=',I3,4X,'IRALPH=',I6,4X,
     1  'NRSHFT=',I3/3X,'IPARCR',4(3X,I7)/
     1  2X,'MRE '9I6/13(10I6/)/)
7913    FORMAT(//)
C*****
C       SPECTRAL DEEMPHASIS
C*****
        Y(1)=YY(1)+RL1-0.4*RL2
        Y(2)=YY(2)+Y(1)-0.4*RL1
        DO 1500 J=3,LTH
1500    Y(J)=YY(J)+Y(J-1)-0.4*Y(J-2)
        RL1=Y(LTH)
        RL2=Y(LTH-1)
C*****
C       WRITE OUT THE DATA
C*****
```

```
      DO 402 J=1,LTH
402   NOUT(J)=Y(J)
      CALL TAPE1(2)
C*****
C     THIS SHIFTS THE DATA FOR NEXT TIME
C*****
      DO 403 J=1,LTH
      JJ=LTH+J
      F(J)=F(JJ)
      IRIN2(J)=IRIN2(JJ)
403   IY2(J)=IY2(JJ)
      GO TO 1000
C*****
C     WRITES THE END OF FILE
C*****
5000  DO 5001 I=1,LTH
5001  NOUT(I)=0
      DO 5002 I=1,32
5002  CALL TAPE1(2)
      NSKIP=NSKIPS
      NEND=0
      IST=1
      CALL TAPE1(6)
      DO 5005 I=1,ICOUNT
      CALL TAPE1(1)
      IF(NEND.NE.0) GO TO 5077
      DO 5006 J=1,LTH
5006  NOUT(J)=NIN(J)
5005  CALL TAPE1(2)
5077  DO 5007 I=1,LTH
5007  NOUT(I)=0
      DO 5008 I=1,8
5008  CALL TAPE1(2)
      CALL TAPE1(5)
      WRITE(6,5009)
5009  FORMAT(1X,'DONE'/)
      END
```


Figure A-3: FORTRAN Listing of Main Program of APCQ Coder

TABLE A-1. VARIABLE NAMES USED IN MAIN PROGRAM OF APCQ CODER (Cont.)

SUBROUTINES	MEANING
FUNCTIONS	
IADD (I1,I2,IOVR)	Add I1 to I2, increment IOVR if overflow
IFMUL (I1,I2)	Fractionally multiply I1, I2; round result
IFDIV (I1,I2)	Fractionally divide I1 by I2
IIMUL (I1,I2,IOVR)	Integer multiply I1 by I2; increment IOVR if overflow
VARIABLES	
LEVLTI, LEVLR1	Error quantizer level at transmitter and receiver
LP	Maximum value of quantizer
IR1N1 IY1	Coefficient predictor filter memory and transmitter receiver
NTOTI	Tape handler data
NTUPS	Tape handler data
NTOTO	Tape handler data
NSKIP	Tape handler data
IST	Tape handler data
NSHFT, NRSHFT	Present normalizer shift
LSHFT, LSHRC	Past frame normalizer shift
NIN	Speech input array
F	Input array after pre-emphasis
ISPIN	Input array after normalization
IV	Reduced waveform
IRIN2, IY2	Pitch filter history (reconstructed waveform)
IA	Predictor coefficients
IR	Autocorrelation values
M	Pitch delay
IALPHA, IRALPH	Pitch Gain Alpha
MALPHA	Quantized Alpha

TABLE A-1. VARIABLE NAMES USED IN MAIN PROGRAM OF THE APCQ CODER (Cont.)

VARIABLES	MEANING
MIE, MRE	Quantized error signal array
IPARCT, IPARCR	PARCOR parameters
ITRAN, IEREC	Coded vector of error signal
YY	Output array before de-emphasis
Y	Output array after de-emphasis
NOUT	Output array to tape

A-2. QUANTIZATION OF TRANSMISSION PARAMETERS

Table A-2 shows the quantization of the filter parameters, the normalization, and the error signal.

TABLE A-2. NUMBER OF TRANSMISSION BITS FOR EACH PARAMETER

Parameter	No. of Bits
pitch gain	3/frame
pitch	6/frame
normalization	3/frame
PARCOR	12/frame
error	2.1/sample

The quantization and dequantization of the pitch gain ALPHA is performed by the subroutines IALPQ and IALPD respectively. Quantization and dequantization of the PARCOR coefficients is performed by subroutines IPARQ and IPARDD respectively, with each PARCOR quantized to 4 bits. Quantization of the pitch is performed simply by subtracting 15 from the calculated value and quantization of the normalization by subtracting 1 from the unquantized value. The quantized values then run from 0 - 60 for pitch (6 bits) and 0 - 7 (3 bits) for normalization. These numbers represent coded values for transmission.

A.3 QUANTIZATION OF THE ERROR SIGNAL

Quantization of the error signal is accomplished by means of an eight-level adaptive quantizer as described previously in Section 2.8.2. This quantizer technique was developed by Jayant as described in the IEEE Proceedings of May 1974.

Figure A-4 shows a flow chart of the subroutine EDQNT-3 that is used to implement the adaptive quantization process. This routine converts the error signal into one of eight numerical values, and adjusts the quantizer step size in accordance with the "Jayant algorithm".

Table A-3 shows the variable names and their meanings as used in subroutine EDQNT-3. Figure A-5 is a FORTRAN listing of this subroutine.

TABLE A-3. VARIABLE NAMES USED IN SUBROUTINE EDQNT3

VARIABLE	MEANING
LP	Maximum value of quantizer level
MIE	Input unquantized error and output quantized value
LEVEL	Present quantizer level

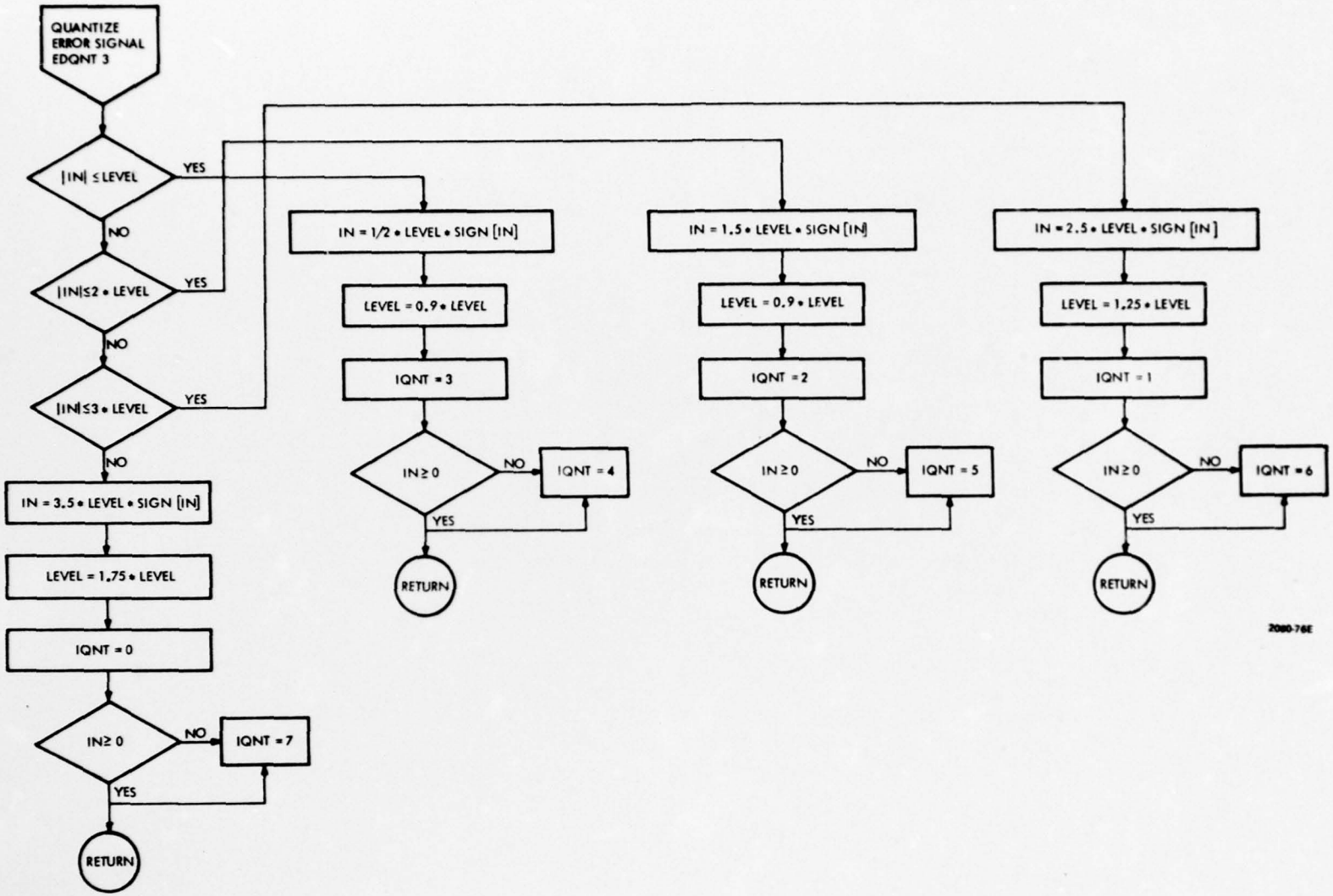


Figure A-4. Flowchart of Subroutine EDQNT3 - Quantizing Error Signal (8-Level Jayant Quantizer)

```
      SUBROUTINE EDQNT3(IN,IQNT,LEVEL)
C     ERROR QUANTIZER AT THE TRANSMITTER
C     EIGHT LEVEL ADAPTIVE QUANTIZER
C     WITH COEFFICIENTS AS PER JAYANT
      DIMENSION COEF(4)
      DATA LP/1024/
      DATA COEF/0.9,0.9,1.25,1.75/
      ISIGN=1
      IF(IN.GE.0) GO TO 10
      IN=-IN
      ISIGN=-1
10    IF(IN.LE.LEVEL) GO TO 1
      IF(IN.LE.2*LEVEL) GO TO 2
      IF(IN.LE.3*LEVEL) GO TO 3
C     TOP MOST QUANTIZER LEVEL
      IQNT=0
      IF(ISIGN.EQ.1) IQNT=7
      IN=3.5*LEVEL*ISIGN
      LEVEL=LEVEL*COEF(4)
      IF(LEVEL.GT.LP) LEVEL=LP
      RETURN
C     SECOND FROM TOP LEVEL
3     IQNT=1
      IF(ISIGN.EQ.1) IQNT=6
      IN=2.5*LEVEL*ISIGN
      LEVEL=LEVEL*COEF(3)
      IF(LEVEL.GT.LP) LEVEL=LP
      RETURN
C     SECOND LEVEL FROM THE BOTTOM
2     IQNT=2
      IF(ISIGN.EQ.1) IQNT=5
      IN=1.5*LEVEL*ISIGN
      LEVEL=LEVEL*COEF(2)
      IF(LEVEL.EQ.0) LEVEL=1
      RETURN
C     BOTTOM LEVEL
1     IQNT=3
      IF(ISIGN.EQ.1) IQNT=4
      IN=LEVEL/2*ISIGN
      LEVEL=LEVEL*COEF(1)
      IF(LEVEL.EQ.0) LEVEL=1
      RETURN
      END
```

Figure A-5: FORTRAN Listing of Error Signal Quantizer

A.4 Dequantization of the Error Signal

The flow chart of Figure A-6 shows the dequantization of the error signal via subroutine EDDNT3. This algorithm performs the inverse operations to those performed by the quantizer at the analyzer and yields as output the sequence of quantizer error levels generated in the analyzer.

Table A-4 contains the variable names and their meanings used in this subroutine. A FORTRAN listing of this subroutine is contained in Figure A-7.

TABLE A-4. VARIABLE NAMES USED IN SUBROUTINE EDDNT3

VARIABLE	MEANING
LP	Maximum value of quantizer level
MIE	Input quantized value
ISIG	Output dequantized value

A.5 Other Subroutines

A.5.1 Introduction

This section presents various subroutines that have not been discussed previously. Table A-5 presents a list of the names of these subroutines and their functions.

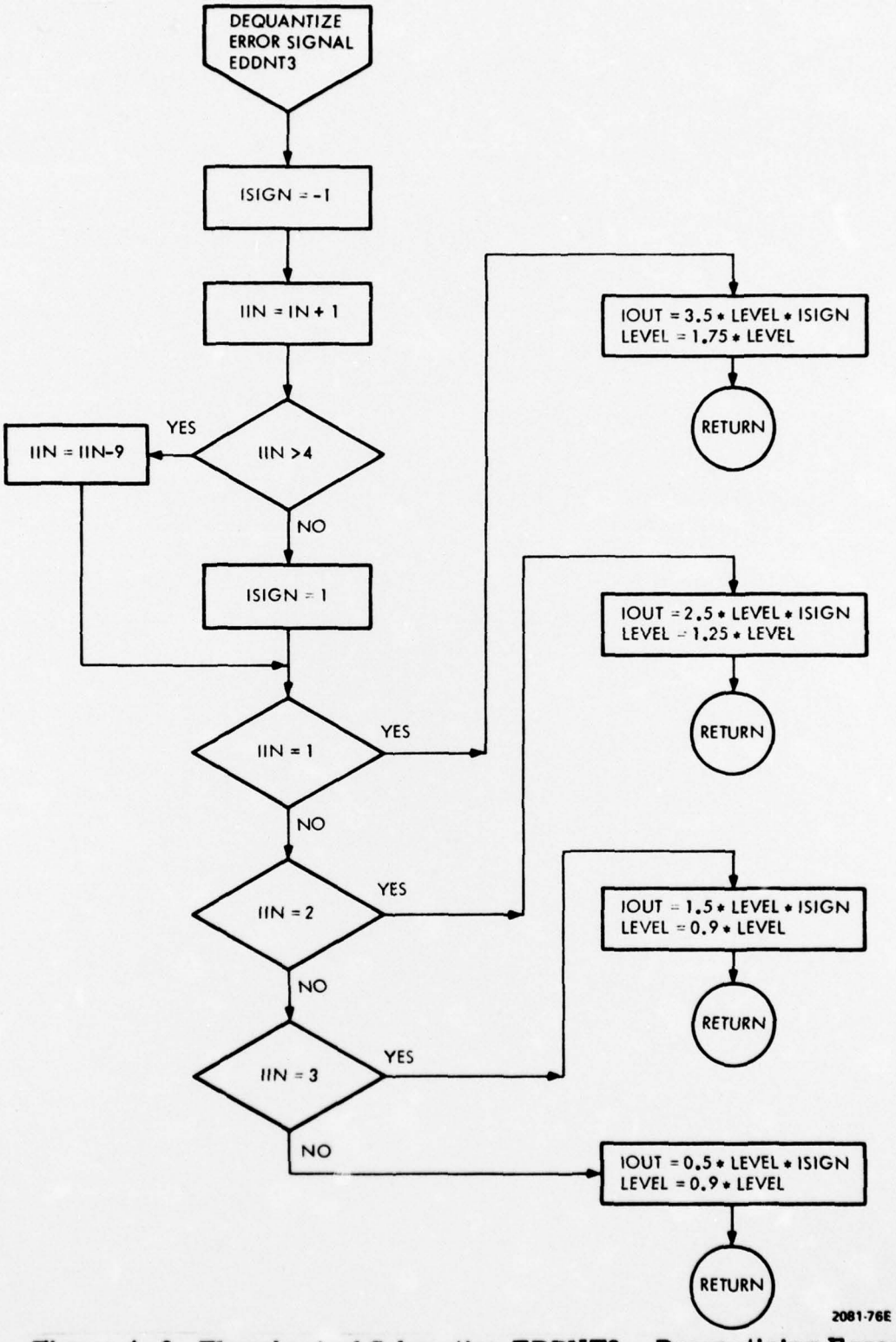


Figure A-6. Flowchart of Subroutine EDDNT3 - Dequantizing Error Signal (8-Level Jayant Quantizer)

```
C
C
      SUBROUTINE EDDNT3(IN,IOUT,LEVEL)
C     SUBROUTINE FOR DEQUANTIZING 3 BIT ADAPTIVE
C     QUANTIZER SIGNAL AT THE RECEIVER
C     FOLLOWS THE JAYANT ALGORITHM
C
C
      DIMENSION COEF(4)
      DATA COEF/1.75,1.25,0.9,0.9/
      DATA LP/1024/
      ISIGN=-1
      IIN=IN+1
      IF(IIN.LT.1.OR.IIN.GT.8) IIN=1
      IF(IIN.LE.4) GO TO 10
      ISIGN=1
      IIN=9-IIN
10    GO TO (1,2,3,4)IIN
C
1     IOUT=3.5*LEVEL*ISIGN
11    LEVEL=LEVEL*COEF(IIN)
      IF(LEVEL.GT.LP) LEVEL=LP
      RETURN
2     IOUT=2.5*LEVEL*ISIGN
      GO TO 11
3     IOUT=1.5*LEVEL*ISIGN
33    LEVEL=LEVEL*COEF(IIN)
      IF(LEVEL.EQ.0) LEVEL=1
      RETURN
4     IOUT=0.5*LEVEL*ISIGN
      GO TO 33
      END
```

Figure A-7: FORTRAN Listing of Error Signal Dequantizer

TABLE A-5. SUBROUTINES AND THEIR FUNCTIONS

SUBROUTINE NAME	FUNCTION
IALPQ	Quantize Alpha To 3 Bits
IALPD	Dequantize Alpha
IPARQ	Quantize PARCOR coefficient
IPARDD	Dequantize PARCOR coefficient
RANERR	Generate and Introduce Random Errors
NORML1	Normalize Input Speech
IADD	Fixed-Point Addition With Overflow Indication
ISUB	Fixed-Point Subtraction With Overflow Indication
IFMUL	Fixed-Point Fraction Multiply With Rounding
ITMUL	Fixed-Point Integer Multiply With Overflow Indication
IFDIV	Fixed-Point Fractional Divide

A.5.2 Subroutines IALPQ and IALPD

Subroutines IALPQ and IALPD each contain one argument. In IALPQ the argument upon entry is the unquantized ALPHA. The subroutine converts this argument to a three-bit representation having values 0 through 7. These values are in a form suitable for transmission. Subroutine IALPD accepts the corresponding received three-bit value and converts it into one of eight possible values for ALPHA by a table look-up procedure.

Figure A-8 contains a FORTRAN listing of both IALPQ and IALPD.

```
C       PROGRAM FOR QUANTIZING ALPHA AND DEQUANTIZING ALPHA
        SUBROUTINE IALPQ(J)
        IF(J.LT.0) J=-J
        IF(J.GT.32250)GO TO 1
        IF(J.GT.31550)GOTO 2
        IF(J.GT.30650) GO TO 3
        IF(J.GT.29050)GO TO 4
        IF(J.GT.25850) GO TO 5
        IF(J.GT.20400) GO TO 6
        IF(J.GT.11850) GO TO 7
        J=7
        RETURN
1       J=0
        RETURN
2       J=1
        RETURN
3       J=2
        RETURN
4       J=3
        RETURN
5       J=4
        RETURN
6       J=5
        RETURN
7       J=6
        RETURN
        END
C       PROGRAM FOR DEQUANTIZING ALPHA
        SUBROUTINE IALPD(J)
        DIMENSION IALP(8)
        DATA IALP/32250,31751,30951,29701,27301,22976,15976,5850/
        J=J+1
        J=IALP(J)
        RETURN
        END
```


Figure A-8: FORTRAN Listing of Subroutines IALPQ and IALPD

A.5.3 Subroutine IPARQ and IPARDD

Subroutines IPARQ and IPARDD quantize and dequantize the PARCOR coefficients. Each subroutine takes one agreement and returns the answer in this same variable. In IPARQ the argument upon entry is the i^{th} unquantized PARCOR(1). The subroutine converts this in a 4 bit representation having values 0 through 15. These values are in a form suitable for transmission. Subroutine IPARDD accepts the corresponding 4 bit value and converts it into one of 16 possible values for the i^{th} PARCOR coefficient. Figure A-9 contains a FORTRAN listing of both IPARQ and IPARDD.

```
      SUBROUTINE IPARQ(J)
      J=J/2**11
      IFLAG=0
      IF(J.GE.0) GO TO 1
      J=IABS(J)
      J=J+16
1     RETURN
      END


      SUBROUTINE IPARDD(J)
      DIMENSION IPD(16)
      DATA IPD/1200,3100,5075,7025,9325,11200,13250,
     1 15225,17500,19475,21450,23400,25700,27075,29550,31500/
      J=J+1
      IFLAG=0
      IF(J.LE.16) GO TO 1
      IFLAG=1
      J=J-16
1     J=IPD(J)
      IF(IFLAG.EQ.1) J=-J
      RETURN
      END
```

Figure A-9: FORTRAN Listing of PARCOR Quantizer (IPARQ) and of PARCOR Dequantizer (IPARDD)

A.5.4 Subroutines RANERR and IXOR

Subroutine RANERR is entered with six arguments for introducing channel errors into the transmission parameters. These arguments include the transmission data, the probability of channel bit error, states for the random number generator, and a count of the total number of errors. The assembly language program IXOR is used in conjunction with RANERR to exclusively "OR" the transmission data with the random bit error data generator within RANERR. Documentation of subroutines RANERR and IXOR are contained in Table A-6 and Figure A-10. Table A-6 lists the variable names contained in subroutine RANERR and their meanings as used in this program. Figure A-10 contains listing of the FORTRAN subroutine RANERR and the assembly language subroutine IXOR.

TABLE A-6. VARIABLE NAMES USED IN SUBROUTINE RANERR

VARIABLE	MEANING
MASK	Mask for introducing a bit of error in either first through 8 bit of word
RANDU	Subroutine for obtaining random number YRN between 0 and 1
PROB	Probability of error any bit
IXOR	Function for exclusive OR in bit of error
IWORD	Output word with errors introduced

```
      SUBROUTINE RANERR(IWORD,KEYM,PROB,IRN,JRN,KOUNT)
C  IWORD - THE WORD CONTAINING THE QUANTIZED PARAMETER
C KEYM - WHICH BIT OF IWORD THE SUBROUTINE WILL OPERATE ON
C PROB - THE FRACTION OF ERRORS
C IRN AND JRN - NOS. USED BY THE RANDOM NO. GENERATOR
C KOUNT - A RUNNING TOTAL OF THE ERRORS IN THIS FRAME
C
C A RANDOM NUMBER BETWEEN 0 AND 1 IS GENERATED AND IF IT
C IS LESS THAN PROB, IWORD IS EXCLUSIVE OR-ED WITH THE
C APPROPRIATE MASK IN ORDER TO FLIP THE BIT POINTED TO BY KEYM.
C
C RANERR IS CALLED ONCE FOR EACH SIGNIFICANT BIT IN IWORD.
C I.E. IF A PARAMETER IS QUANTIZED TO 4 BITS, RANERR
C IS CALLED WITH KEYM EQUAL TO 1, TO 2, TO 3, AND TO 4.
C
      DIMENSION MASK(8)
      DATA MASK /1,2,4,8,16,32,64,128/
      CALL RANDU(IRN,JRN,YRN)
      IF (YRN-PROB) 200,500,500
200   JUNK=IXOR(IWORD,MASK(KEYM))
      IWORD=JUNK
      KOUNT=KOUNT+1
500   RETURN
      END
```

```
      .TITLE IXOR
      .GLOBL IXOR
;THIS ASSEMBLY LANGUAGE SUBROUTINE TAKES EXCLUSIVE OR OF
;TWO INPUT ARGUEMENTS AND LEAVES ANSWER IN REGISTER 0 FOR A FORTRAN
;FUNCTION CALL
R0=%0
R1=%1
R5=%5
;
IXOR:  TST (R5)+              ;JUMP OVER BRANCH
       MOV @(R5)+,R0          ;GET FIRST ARGUEMENT
       MOV @(R5)+,R1          ;GET SECOND ARGUEMENT
       XOR R1,R0              ;EXCLUSIVE OR WITH ANSWER IN R0
       RTS R5                 ;LEAVE
       .END
```

Figure A-10: FORTRAN Listing of Subroutine RANERR and Assembly Listing of IXOR

A.5.5 Subroutine NORML1

Subroutine NORML1 contains five arguments, the unnormalized input array, the normalized output array, the number of data points in the array, the value of the largest point after normalization, and the number of shifts computed by the subroutine. An assembly language listing of this subroutine is contained in Figure A-11.

A.5.6 Function IADD, ISUB, IFMUL, IIMUL and IFDIV

The routines discussed in this section are called "Functions" since they return an argument to the main program via a specified register rather than one of the calling arguments. Functions IADD and ISUB arguments each contain three. The first two are the input numbers to be added (or subtracted) while the third argument is incremented if an overflow occurs. The function IFMUL has two arguments and returns the upper sixteen bits of the product (with rounding) of these two arguments to the calling program. This function is sometimes referred to as a fractional multiply because it preserves the location of the binary point.

The function IIMUL has three arguments which are the two input numbers to be multiplied and the overflow indicator. This function returns the lower 16 bits of the 32-bit product and is referred to as an integer multiply.

The function IFDIV accepts two inputs and returns the fractional part of the division to the calling program.

Assembly listing of these fixed point arithmetic functions are given in Figure A-12.

```
        .TITLE NORMLX.PAL
;ROUTINE NORMALIZES(BY SHIFTING TO THE LEFT)AN ARRAY OF INTEGER
;NUMBERS OBTAINED FROM A FORTRAN ARRAY AND PLACES THE RESULT IN A
;NEW ARRAY WITH ONE WORD PER POINT INSTEAD OF THE TWO WORDS
;ALLOWED IN FORTRAN
;
;       CALL NORML(IN,OUT,N,NSPEC,NS)
;
;       IN=FORTRAN INPUT ARRAY
;       OUT=OUTPUT ARRAY(ONE WORD/INTEGER)
;       N=# OF WORDS IN ARRAY
;       NSPEC=SPECIFICATION OF SHIFTED OUTPUT
;       NS=# OF SHIFT COMPUTED BY NORM
                .GLOBL NORML1
        R0=%0
        R1=%1
        R2=%2
        R3=%3
        R4=%4
        R5=%5
;
NORML1: TST(R5)+        ;SET OVER FORTRN BRANCH
        MOV R0,R0S      ;SAVE GENERAL REGISTERS 0,1,2,3,4,&5
        MOV R1,R1S
        MOV R2,R2S
        MOV R3,R3S
        MOV R4,R4S
        MOV (R5)+,R0    ;GET ADRESS OF INPUT
        MOV R0,INSAV    ;SAVE INPUT ADRESS
        MOV (R5)+,R1    ;OUTPUT ADRESS
        MOV @(R5)+,R2   ;# OF DATA WORDS
        MOV R2,N        ;SAVE # OF WORDS IN ARRAY
        MOV @(R5)+,R3   ;SAVE LOCATION OF SHIFTED VALUE
        MOV R3,NSPEC
        CLR R4          ;INCLUSIVE OR
LOOP:   MOV (R0)+,R3    ;GET INPUT POINT
        BPL PLUS
        NEG R3          ;CAN'T BIS NEG. #'S
PLUS:   BIS R3,R4       ;R4 CONTAINS SUM OF INCLUSIVE OR
        TST (R0)+       ;TWO WORD INTEGER
        SOB R2,LOOP
        CLR R3          ;SET UP TO COUNT # OF SHIFTS
        TST R4
        BEQ ALL         ;PREVENTS CONTINUOUS LOOPING ONE ALL 0'S
LOOP1:  INC R3
        ASL R4          ;SHIFT LEFT ONCE
        BVC LOOP1       ;CHECK FOR OVERFLOW
                        ;# OF LEFT SHIFTS REQUIRED
ALL:    MOV N,R2
                        ;NORMALIZE INPUT
        MOV #15.,R0     ;SPECIFY LOC. OF DEC.POINT
        SUB NSPEC,R0
        SUB R0,R3
        MOV INSAV,R0
LOOP2:  MOV (R0)+,R4    ;GET INPUT
        TST (R0)+       ;2ND WORD INTEGER
        ASH R3,R4
        MOV R4,(R1)+    ;DEPOSIT INTO OUTPUT
        TST (R1)+
        SOB R2,LOOP2
        MOV R3,@(R5)+   ;RETURN SHIFT COUNT
        MOV R0S,R0
        MOV R1S,R1
        MOV R2S,R2
        MOV R3S,R3
        MOV R4S,R4
        RTS R5          ;RETURN
;BUFFER STORAGE
R0S:    .WORD 0         ;GENERAL REGISTER TEMPORARY STORAGE
R1S:    .WORD 0
R2S:    .WORD 0
R3S:    .WORD 0
R4S:    .WORD 0
N:      .WORD 0         ;LENGH OF INPUT ARRAY
INSAV:  .WORD 0         ;ADRESS OF INPUT ARRAY
NSPEC:  .WORD 0         ;SPEC. OF SHIFTED OUTPUT
        .END
```

Figure A-11: Assembly Language Listing of NORML1

```
        .TITLE IARITH.MAC
;
;THIS FUNCTION ADDS TWO NUMBERS AND INCREMENTS THE
;THIRD ARGUMENT IF THERE IS AN OVERFLOW
;SAME FOR THE SUBTRACT CASE
;
        R0=%0
        R1=%1
        R5=%5
        .GLOBL  IADD,ISUB,IFMUL,IFDIV,IIMUL
;A=IADD(I,J,K)
;IA=I+J
;K=K+1 IF OVERFLOW
;
IADD:   TST (R5)+               ;GET OVER BRANCH
        MOV @(R5)+,R0
        ADD @(R5)+,R0           ;ADD
        BVS A                   ;JUMP ON OVERFLOW
        TST (R5)+               ;GET OVER ARGUMENT
        RTS R5
A:      INC @(R5)+
        RTS R5
;
ISUB:   TST (R5)+
        MOV @(R5)+,R0
        SUB @(R5)+,R0
        BVS A
        TST (R5)+
        RTS R5
;THIS SUBROUTINE MULTIPLIES TWO NUMBER IN FRACTIONAL FORM WITH ROUNDING
;AS IN THE GTE SYLVANIA PSP INSTRUCTION MLY
IFMUL:  TST (R5)+
        MOV @(R5)+,R0           ;GET THE FIRST NUMBER
        MUL @(R5)+,R0           ;MULTIPLIY BY THE SECOND(32 BIT ANSWER)
        ASHC #1,R0              ;ELIMINATE ONE SIGN BIT
        TST R1                  ;ROUND IF THE 17TH BIT IS A ONE
        BGE NRND                ;NO ROUNDING NECESSARY
        INC R0                  ;ROUNTING OCCURS IF R1 IS NEGATIVE
NRND:   RTS R5
;FUNCTION IFDIV(DIVIDEND,DIVISOR)
;FRACTIONAL DIVIDE
;
IFDIV:  TST(R5)+
        MOV @(R5)+,R0           ;GET DIVIDEND
        CLR R1
        ASHC #-1,R0             ;ADJUST FOR DIVIDE
        DIV @(R5)+,R0           ;DIVIDEND/DIVISOR
        RTS R5                  ;RETURN
;
;FUNCTION IIMUL
;INTEGER MULTIPLY
;IMUL(A,B,C)
;ANSWER = A*B WITH C INCREMENTED IF OVERFLOW
;
IIMUL:  TST (R5)+
        MOV @(R5)+,R0           ;GET FIRST NUMBER
        CLC                     ;CLEAR THE CARRY FLAG
        MUL @(R5)+,R0           ;MULTIPLY THE NUMBERS
        MOV R1,R0               ;MOV INTEGER PART TO R0
        BCS A                   ; BRANCH IF OVERFLOW
        TST (R5)+
        RTS R5
        .END
```

Figure A-12: Assembly Listing of Fixed-Point Arithmetic Functions